

SÁLYI-SZELEZSÁN

**ADAT-
BÁZISOK**



Sályi János — Szelezsán János

ADATBÁZISOK

**Nyitott rendszerű képzés
– távoktatás – oktatási segédlete**

TANKÖNYV



**Gábor Dénes
Műszaki Informatikai
Főiskola**

Sályi János - dr. Szelezsán János

ISBN 963 577 124 X

Kiadó: LSI Oktatóközpont

Felelős vezető: Dr. Kovács Magda

Témafelelős: Flier István

LIGATURA KFT - VÁCI ÁFÉSZ NYOMDA
Munkaszám: 354-94

ELŐSZÓ

Az adatbázisok fontos területét jelentik a számítástechnikai alkalmazásoknak. Az alkalmazói szoftverek közül kiemelkedő szerepet kaptak az adatbáziskezelő rendszerek. A programozók, szervezők jelentős részének mindennapi munkájában szüksége van az adatbázisokkal, illetve az adatbáziskezelő rendszerekkel kapcsolatos ismeretekre.

Könyvünkben rövid áttekintést adunk a legfontosabb tudnivalókról. Az első fejezetben ismertetjük az adatmodellek alapelemeit és a három adatbázismodell lényegét. A második fejezetben a relációs adatmodellt tárgyaljuk, a harmadik fejezetben a szabvánnyá vált SQL nyelvet ismertetjük. A negyedik fejezetben a hálós adatbázisokról és – igazán csak áttekintés jelleggel – az ilyen adatbázisokat kezelő IDMS nyelvről szólunk.

A szerzők

1. FEJEZET

ADATMODELLEK, ADATBÁZISOK

1.1. Bevezetés

Az első számítógépeket matematikai feladatok megoldására készítették. Azokra az alkalmazásokra, amelyeknek ma vagyunk tanúi, Neumann János (a számítógépek „atyja”) talán egyáltalán nem gondolt.

Már a 60-as évek elején a számítógépek alkalmazásának nagyobbik részét az ún. „adatfeldolgozás” tette ki. Természetesen a matematikai jellegű alkalmazásokban is adatokkal végez a gép műveleteket, de ebben az esetben a valóságos jelenségek paraméterei (adatai) közötti összefüggést, kapcsolatot leíró modell valamilyen matematikai struktúra. Az ilyen feladatok megoldása általában egy matematikai eljárás végrehajtását jelenti.

Természetesen az egyszerű adatfeldolgozási feladatok megoldása is jelenthet matematikai jellegű eljárást, algoritmust.

Korán rájöttek a szakemberek arra, hogy az „egyszerű” adatfeldolgozás is jobban „gépesíthető”, ha az adatok közötti akár egyszerű kapcsolatokat struktúrának tekintjük, és *adatmodellekben* gondolkodunk. Ez lehetővé teszi az egységes kezelést, olyan nyelvek kialakítását, amelyekkel szabványosan kezelhetők a tartalmuk szempontjából teljesen eltérő adathalmazok is. Az olyan adathalmazokat, amelyeket modellbe foglalva kezeltek, *adatbankoknak*, később pedig *adatbázisoknak* nevezték el. Adatbázisokon voltaképpen adatoknak kapcsolataikkal együtt való ábrázolását, tárolását értjük.

Annak, hogy egy konkrét adatrendszer egy megadott szerkezetű adatbázisba foglaltak egybe, nagyon nagy előnye, hogy egyszerre sokan használhatják. Maguk az adatok csak „egyszeresen” vannak tárolva, a közöttük lévő kapcsolatok (az adatrendszer-struktúra) ismerete mégis lehetővé teszi, hogy a felhasználók egymástól függetlenül hozzáférjenek szinte úgy, hogy ki-ki „kiszakít” magának egy részt (nem fizikailag), vagy átszerkeszti az adathalmaz egy részét a maga számára (ezt az angol *view* szóból felhasználói „nézet”-nek nevezzük). Az adatbázisok tehát „többfelhasználós” adathalmazok.

Adatbázisok modelljéül szolgáló szerkezet kidolgozására számos próbálkozás történt. Ezek a próbálkozások az alábbi három fő irány körül csoportosultak.

1. A *formalista megközelítés* irányzatának képviselői gráfokkal, diagramokkal kísérelték meg felírni az adatok közötti kapcsolatot, a szerkezetet (Bachmann, Nijssen, Sowa stb. modelljei). Egy kis erőltetéssel azt is mondhatjuk, hogy ezek a később elterjedt *hierarchikus*, illetve *hálós adatmodell* különféle változatait képezték.
2. A *szemantikai megközelítés* lényege az volt, hogy a modellt teljesen verbálisan, szavak segítségével kísérelték meg leírni. Ez a modellfajta teljesen életképtelen volt.

3. A *matematikai* jellegű modellekben az adatok közötti kapcsolatot szintaktikusan pontosan, matematikai struktúrák felhasználásával írták le. Több egymástól eltérő felfogásban próbálkoztak, leginkább Codd *relációs adatmodellje* (1970) látszott életképesnek, de azt senki sem gondolta, hogy 1990-ben az adatbáziskezelő rendszerek nagy része a Codd-féle modellre támaszkodik.

1.2. Adatbáziskezelő rendszerek

Az adatbázis önmagában semmit sem ér. Akkor válik értékké, ha megadunk hozzá egy olyan szoftvert (programnyelvet), amellyel az adatbázist kezelni tudjuk. Az ilyen szoftver nagyon bonyolult, általában több elemből álló rendszert képez. Ezt a szoftvert *adatbáziskezelő rendszernek* nevezzük.

Az adatbázisokkal lényegében két fontos műveletet kell elvégezni: az adatbázis *létrehozását* (az adatbázis szerkezetének definiálását és a szerkezet feltöltését konkrét adatokkal), módosítását, valamint adatok *visszakeresését*, lekérdezését az adatbázisból. Ennek megfelelően az adatbáziskezelő rendszerek is két fő elemből állnak: az egyik rész az *adatdefiníciós nyelv*, a másik pedig a *lekérdező*, illetve adatkezelő nyelv.

Az adatbáziskezelő rendszereknek mint nyelveknek két fajtája van: az egyik a *beépülő típus*, amelyik voltaképpen nem is igazi programnyelv, hiszen csak egy másik valódi programnyelvvvel (Cobol, PL1, Pascal stb.) együtt használható fel alkalmazói programok írására, alkalmazói rendszerek felépítésére. Erre példa a könyvünkben szereplő IDMS és SQL nyelv. A másik típus valóságos programozási nyelv. Ezeknél ahhoz, hogy alkalmazói rendszereket készítsünk az adatbázis felhasználására, nem kell más programnyelvet bekapcsolni. Erre példa a dBase (III, IV) nyelv.

Az adatbáziskezelő rendszerek a fő funkciókon kívül több segédfeladatot is ellátnak. Ezek közül kiemeljük a következőket.

1. *Adatvédelem, adatbiztonság.* Nem minden felhasználónak van joga az adatbázis minden adatához hozzáférni. Amikor a felhasználó programján keresztül az adatbázis-hoz fordul, jelszóval kell azonosítania magát. Ezt az adatbáziskezelő rendszer értékeli, és csak azokhoz az adatokhoz engedi hozzáférni, amelyre jogosultsága van. Az adatok védelme nagy adatbázisoknál rendkívül fontos, hiszen az adatbázis tönkretétele nagyon nagy károkat okozhat (gondoljunk egy óriási bank adatbázisára). Az adatok védelmének szoftvereszközökkel való biztosítása programozói szemszögből nézve igen érdekes, de nagyon nehéz feladat.
2. *Az integritási feltételek teljesülésének figyelése.* Az adatbázis adataival kapcsolatban gyakran meg lehet fogalmazni olyan feltételeket, amelyek ellenőrzésével az adatbázis létrehozásakor kiszűrhetők azok az input adatok, amelyek nem az adott adatbázisba valók (hibások). Az adatbázis belső szerkezete is hordozhat olyan információkat, amelyek meghatározzák, hogy új adatok bevitele esetén ezek illeszkednek-e az adatbázishoz. Például kiköthetjük, hogy a dátum nevű adattétel nem vehet fel az 1900-as évek előtti dátumot, vagy előfordulhat, a rendszer olyan, hogy két adat megegyezése valahol az adatbázisban maga után hozza másik két adat megegyezését. Az ilyen típusú információkat *integritási feltételeknek* nevezzük. Az adatbáziskezelő feladata ezen integritási feltételek teljesülésének vizsgálata is.

3. *Szinkronizáció.* Különösen nagy adatbázisoknál egyidejűleg nagyon sok felhasználó fordulhat esetleg ugyanazon adathoz. Ráadásul úgy, hogy egyik éppen módosítani akarja, a másik pedig lekérdezni. Ezeknek a *holtpont* helyzeteknek a megoldása nagyon nehéz feladat.

1.3. Az adatmodellek alapelemei

Az adatmodelleket az alábbi három elem segítségével adjuk meg, írjuk le.

1.3.1. Az egyed fogalma

Egyednek hívunk minden olyan dolgot (objektumot), ami minden más dologtól (objektumtól) megkülönböztethető. Például: *dolgozó, autó, érzelem, személy.* Az egyed konkrét dolgok absztrakciója, konkrét dolgokat fog egybe. Például az autó mint egyed sok autót jelent. Egy adott egyed által képviselt konkrét elemek halmazát *egyedhalmaznak* nevezzük. Például a *dolgozó* nevű egyed egyedhalmaz az összes dolgozóból áll, a *személy* egyedhalmaz az összes személyt tartalmazza, az *érzelem* egyedhalmaz az összes érzelem. Egy adott valóságos objektum különböző egyedekkel tükrözhető. Egy konkrét vállalat nézőpontunktól függően lehet *vevő, szállító, adós, hitelező* stb. egyed. Egy konkrét személy lehet egy vállalatnál *dolgozó*, de ha kórházban van, akkor *beteg*. Azt, hogy egy konkrét jelenség vagy folyamat esetén mit tekintünk egyednek, voltaképpen az egyedhalmaz konkrét elemei határozzák meg. Például a *személy* nevű egyedhalmaz elemei lehetnek az összes személyek, de a *személy* egyedhalmaz állhat az élő személyekből.

Megjegyzés. A szerzők egy része egyed helyett *egyedtypust* mond, az egyedtypus egy konkrét értékét, tehát az egyedhalmaz egy elemét pedig *előfordulásnak* nevezi. Ebben a felfogásban az egyednek két *szintjét* különböztetjük meg: az első szint (az egyedtypus) az absztrakciós szint, a második pedig az előfordulás szint. (Az egyedtypus szóban a típus nem fajtát, féleséget jelent, hanem azt, hogy az egyed egy absztrakció, ami sok konkrét dolgot képvisel, nevezetesen éppen az egyedtypushoz tartozó előfordulásokat.)

1.3.2. A tulajdonság fogalma

Az egyedeket *tulajdonságokkal (attribútumokkal)* írjuk le. A dolgozó nevű egyed tulajdonságai lehetnek például: *név, besorolási kulcsszám, fizetés, adólevonás* stb. Az autó egyedtypus tulajdonságai lehetnek például: a *típus, a rendszám, a szín, a súly* stb. A tulajdonság is voltaképpen egy absztrakció, amely több elemet foglal egybe, tehát konkrét értékekből áll. Például a *név* tulajdonság konkrét értékei lehetnek: Kovács János, Nagy Pál, Kiss János stb. Az autó egyedtypusnál a *rendszám* tulajdonság értékei lehetnek: ABA-127, NHZ-943, ZYX-122 stb.

Vegyük észre, hogy a tulajdonság értékeivel voltaképpen egy adott egyed konkrét értékét (tehát az egyedhalmaz elemét) határozzuk meg. Ha megadjuk egy dolgozó *nevét, besorolási kódszámát, fizetését* stb., ezzel kijelöltük a dolgozó egyed egy konkrét

értékét, a dolgozó egyedhalmaz egy elemét (például: Nagy Pál, S-125, 15000). Azt, hogy egy adott egyedet milyen tulajdonságokkal adunk meg, éppen az adatmodell kimunkálásakor határozzuk meg.

Amennyiben egy tulajdonság vagy tulajdonságok egy csoportja egyértelműen meghatározza, hogy az egyed melyik értékéről van szó (vagyis az egyedhalmaz melyik eleméről), akkor ezeket a tulajdonságokat együtt *kulcsnak* nevezzük. Nyilvánvaló, hogy elvileg minden egyednek (egyedhalmaznak) van kulcsa, hiszen az egyedet éppen úgy határoztuk meg, hogy megkülönböztethető egy másik egyedtől. Ha nem tudunk kiválasztani olyan tulajdonságokat, amelyeknek értékei egyértelműen meghatározzák az egyedhalmaz elemeit, akkor bevezetünk egy olyan tulajdonságot, amelynek értékei sor-számok, kódszámok, speciális azonosítók. A dolgozó nevű egyednél a személyi szám mint a dolgozó attribútuma be tudja tölteni a kulcs szerepét, hiszen egyértelműen azonosítja a dolgozókat, vagyis a dolgozó egyedhalmaz elemeit.

Megjegyzés. Ugyanúgy, mint az egyednél, egyes szerzők a tulajdonság helyett is *tulajdonságtípust* mondanak, és ennek megfelelően a tulajdonság konkrét értékeit a *tulajdonságtípus előfordulásainak* nevezik.

1.3.3. A kapcsolat fogalma

Kapcsolatnak nevezzük az egyedek közötti viszonyok fogalmi tükörképeit. A kapcsolat mindig valóságos objektumok közötti viszonyt fejez ki, hiszen az egyed ilyen objektumokat képvisel.

Objektumok közötti viszony például az, hogy a *személyek a vállalatok dolgozói*. Másik példa: ha a *vevő* az egyik egyed, *rendelés* a másik, akkor a vevő és a rendelés egyed típusok között *vevő-rendelése* kapcsolat van, hiszen a vevő rendelést ad le, minden vevőhöz tartoznak rendelések.

Megjegyzés. Kapcsolat helyett is gyakran *kapcsolattípust* mondanak, és a konkrét kapcsolatértékeket a *kapcsolattípus előfordulásainak* nevezik.

Egy kapcsolatértékre példa: ha a *vállalat* és a *személy* egyed típusok közötti *vállalat-személy* kapcsolatot tekintjük, akkor ennek a kapcsolatnak az előfordulása: „Kovács István (a *személy* egyedhalmaz egy értéke) a SZÁMALK (mint a *vállalat* egyedhalmaz egy eleme) dolgozója”.

A kapcsolatok fajtái

A kapcsolat voltaképpen az egyedhalmazok elemei közötti viszonyt fejezi ki. Attól függően, hogy az egyedhalmaz elemei milyen módon kapcsolódnak egymáshoz, három esetet különböztetünk meg.

a) Egy-egy típusú kapcsolat (1:1 kapcsolat)

A kapcsolat lehet olyan, hogy az egyik egyedhalmaz mindegyik eleméhez a másik egyedhalmaznak pontosan egy eleme kapcsolódik (tehát a két egyedhalmaz egymásba kölcsönösen egyértelműen képezhető le). Az ilyen kapcsolatot *egy-egy* (jelben 1:1) *típusúnak* nevezzük. Például a SZEMÉLYI SZÁM mint egyed és a SZEMÉLY egyed között a kapcsolat 1:1 típusú.

b) *Egy-több típusú kapcsolat (1:N kapcsolat)*

Azt mondjuk, hogy az A egyed és a B egyed között *egy-több* (jelben 1:N) *típusú* kapcsolat van, ha az A egyedhalmaz mindegyik eleméhez a B egyedhalmaznak több eleme is tartozhat. Például a MEGYE egyed és a TELEPÜLÉS egyed között 1:N kapcsolat van. (Mindegyik megyéhez több település tartozik; feltesszük, hogy egy település sem tartozik két megyéhez.)

c) *Több-több típusú kapcsolat (N:M kapcsolat)*

Több-több (jelben N:M) *típusú* kapcsolatnak nevezzük az A egyed és a B egyed közötti kapcsolatot, ha az A egyedhalmaz minden eleméhez a B egyedhalmaz több eleme tartozhat, és fordítva egy B-beli elemhez is több A-beli elemet rendelhetünk hozzá. Például legyen az egyik egyed a HÁZAK, a másik a SZÍNEK. Nyilvánvaló, hogy egy ház lehet több színű is, és egy szín lehet több házon is.

Összefoglalásként lássunk most egy együttes példát a fenti fogalmakra.

Legyen a *személy* az egyed. Ekkor a *születési dátum*, az *iskolai végzettség*, a *nem*, a *foglalkozás* tulajdonságok. A kapcsolat a *személy* és a *vállalat* közötti *foglalkoztatási viszony*. Jelöljük ezt a kapcsolatot így: *vállalat-személy*. Ezeknek az absztrakt objektumoknak az értékei konkrét *személyek* (Kiss Péter, Kovács Istvánné, Nagy Gábor stb.), konkrét *születési dátumok* (1953. június 3., 1938. november 14., 1963. február 26. stb.), konkrét *nemek* (férfi, nő), konkrét *foglalkozások* (programozó, takarítónő, közgazdász stb.), és konkrét *foglalkoztatási viszonyok* (pl. a Számalk vállalat dolgozói).

Egy információrendszer tervezése során nyilván csak a számunkra lényeges objektumokat vesszük figyelembe, és az ezeket leíró egyedeket építjük be a modellünkbe. Mint említettük, a különböző célú alkalmazási rendszerekben ugyanazt az objektumot más és más adatokkal, tulajdonságokkal adhatjuk meg. Más jellemzi Kovács Istvánt, a dolgozót, mint Kovács Istvánt, a beteget. Persze a jellemzők egy része a két rendszerben közös is lehet.

Egy adatmodellben valamennyi egyedeket egyenként véges számú tulajdonsággal írunk le. Ezek a tulajdonságok együttesen alkotják az egyedtípus *belső szerkezetét*.

1.4. Az adatmodell és az adatbázisok

Az előzőek alapján egyszerűen megfogalmazható az *adatmodell* fogalma. Az adatmodell egyedek, tulajdonságok és kapcsolatok halmaza, amely absztrakt módon tükrözi a valós objektumoknak, azok jellemzőinek (tulajdonságainak) és viszonyainak (kapcsolatainak) elvont kategóriáit.

Az egyedeket, a tulajdonságokat és a kapcsolatot *adatmodell-elemeknek* nevezzük. Attól függően, hogy ezekből milyen módon hozzuk létre az adatmodell szerkezetét, különféle adatmodelleket kapunk. Mint említettük, a sok próbálkozásból három adatmodell terjedt el az eltelt 30 év alatt. Az alábbiakban röviden ismertetjük ezeket.

1.4.1. A hálós adatmodell

A „történelem” folyamán nem ez volt időben az első adatmodell, de mivel úgy tekinthető, mint a hierarchikus általánosítása (azaz a hierarchikus ennek speciális esete), ezért a hálós adatmodell ismertetésével kezdjük.

A hálós adatmodell szerkezetét gráffal adjuk meg. E gráfban a csomópontok az *egyedek*, az élek pedig a *kapcsolatot* fejezik ki. Az egyedeket *tulajdonságaikkal* írjuk le.

Példaként felírjuk egy szállítási „rendszer” modelljét.

Legyenek az egyedek: VEVŐ, RENDELÉS, TERMÉK, TÉTEL.

Az egyedek tulajdonságait rajzban adjuk meg:

VEVŐ

VEVŐKÓD	VEVŐNÉV	VEVŐCÍM
---------	---------	---------

RENDELÉS

RENDELÉSI-SZÁM	REND-DÁTUM	HATIDŐ
----------------	------------	--------

TERMÉK

TERMÉKKÓD	TERMÉKNÉV	TERMÉKMÉRET	EGYSÉGÁR
-----------	-----------	-------------	----------

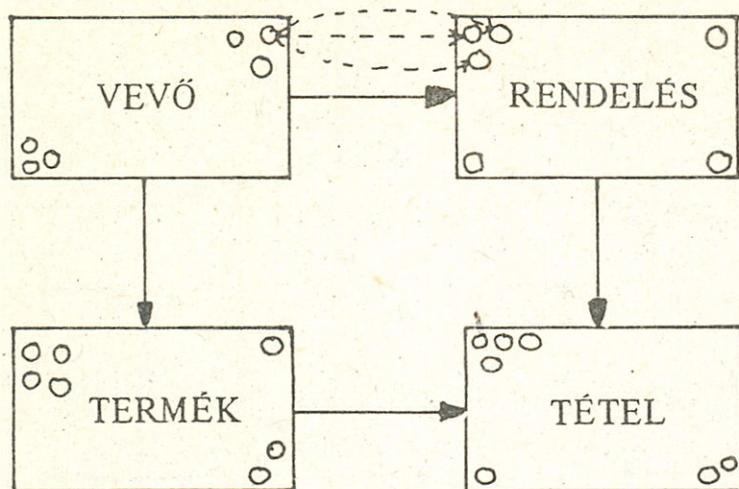
TÉTEL

TÉTELSZÁM	RENDELT-MENNYISÉG	INDEX
-----------	-------------------	-------

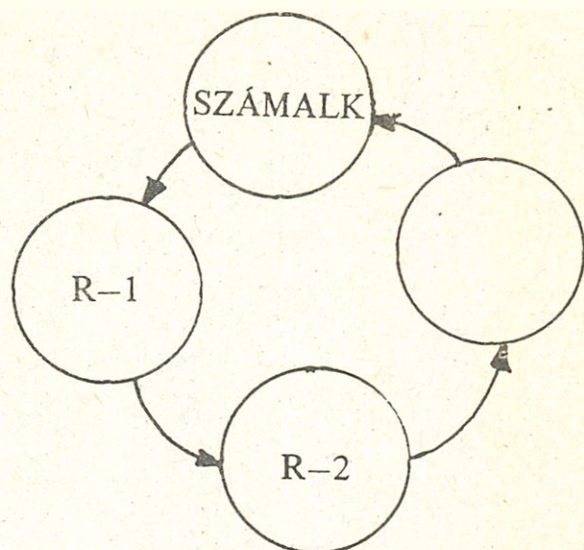
A szállítási modellt az 1.1. ábrán látható gráffal adhatjuk meg.

A gráf VEVŐ csomópontját képező téglalapot úgy foghatjuk fel, hogy tartalmazza az összes vevőt, aki a szállítási modellben szerepel. Ezeket a konkrét vevőket körrel jeöltük a téglalap belsejében. Minden vevőhöz tartozik rendelés (általában több), tehát kapcsolat van köztük; ez a „VEVŐ-RENDELÉS” kapcsolat. A RENDELÉS téglalapban az összes rendelés benne van, mégpedig úgy, hogy egy konkrét vevő rendelései egy csokorba vannak összegyűjtve, pontosabban rá vannak „fűzve” arra a VEVŐ-re, aki ezeket a rendeléseket kiadta. A VEVŐ és a RENDELÉS közötti folytonos nyíl (él) a konkrét VEVŐ-k és a konkrét RENDELÉS-ek között húzott „szaggatott” nyilakat (éleket) jelöli az 1.1. ábrán. Legyen például a VEVŐ egyedhalmaz egyik eleme a SZÁMALK. Ekkor a VEVŐ-RENDELÉS kapcsolat egy konkrét értéke pl. az 1.2. ábrán látható.

Itt R-1, R-2 a SZÁMALK különböző rendeléseit jelenti. Kényelmi okokból nem írtuk fel sem a SZÁMALK-nak, sem a rendeléseinek a tulajdonságait, és a tulajdonságok konkrét értékeit.



1.1. ábra



1.2. ábra

A SZÁMALK a VEVŐ nevű egyed egy értéke, tulajdonságainak értékei legyenek:

VEVŐKÓD: SZ-28
 VEVŐNÉV: SZÁMALK
 VEVŐCÍM: Bp. Szakasits 68.

tehát az 1.2. ábrán a SZÁMALK körben voltaképpen az alábbi szerkezet szerepel:

SZ-28	SZÁMALK	Szakasits 68.
-------	---------	---------------

Ehhez hasonlóan az R-1, vagyis a RENDELÉS egyed értékei közül a SZÁMALK-hoz tartozó első rendelésben például az alábbiak szerepelhetnek:

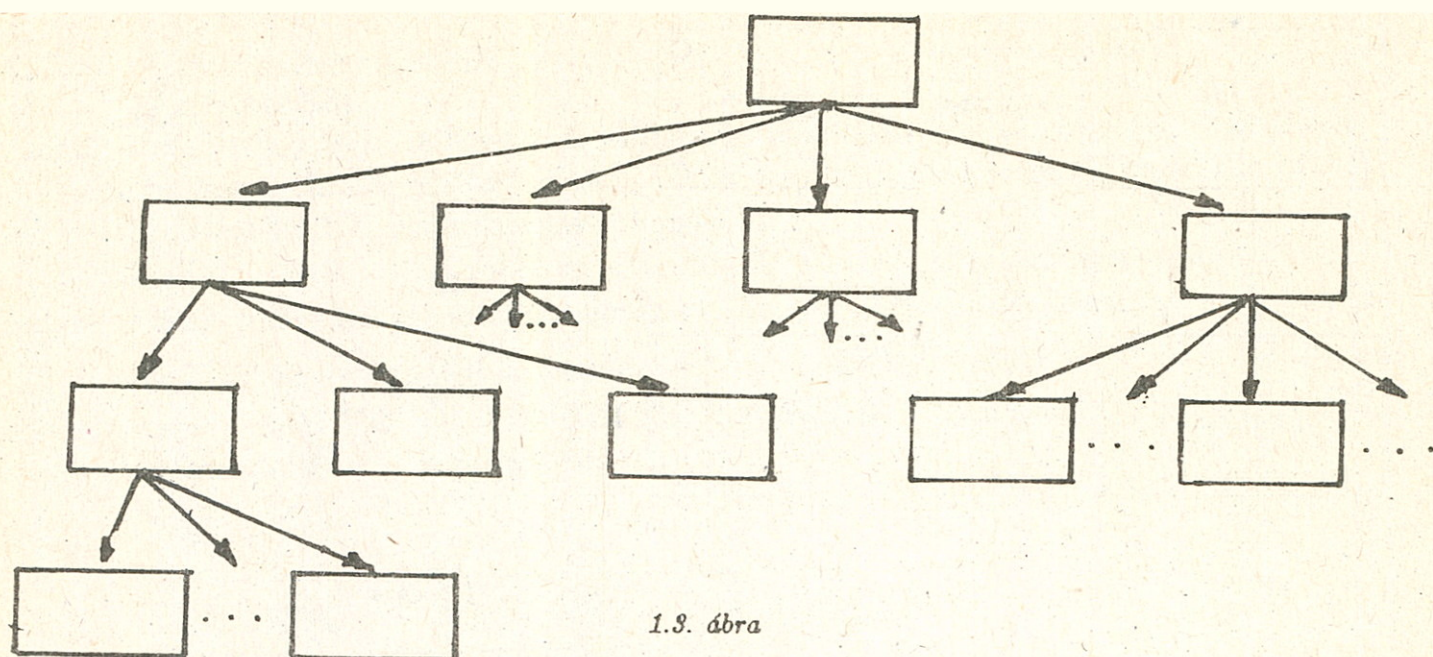
R-812	05.02.91.	12.12.91.
-------	-----------	-----------

Ebből, amit a VEVŐ-RENDELÉS kapcsolatról elmondtunk, az olvasó bizonyára látja, mit jelent a többi nyíl (él) az ábrán szereplő egyedek között.

A gráfból azt is látjuk, hogyan kereshető meg egy konkrét érték az olyan adatbázisokban, amelyeket hálós adatmodellben építettünk fel. Például ha szükségünk volna a SZÁMALK első RENDELÉSÉ-ben szereplő első TÉTEL-re, akkor az alábbi utat kellene bejárni a gráfon: megkeressük a VEVŐ egyed értékei közül a SZÁMALK-ot, majd az él mentén elmegyünk a RENDELÉS egyedhez (amelyben a SZÁMALK-hoz tartozó rendelések egybe vannak gyűjtve), és itt megkeressük a SZÁMALK első rendelését, majd (az él mentén) tovább megyünk az ehhez a rendeléshez tartozó TÉTEL-hez.

1.4.2. A hierarchikus adatmodell

A hierarchikus adatmodell szerkezetét szintén gráffal adjuk meg (amelyben a csomópontok egyedek, a kapcsolatokat pedig az élek fejezik ki), de a gráf speciális, nevezetesen *fa*. Egy hierarchikus adatmodell szerkezetét az 1.3. ábrán láthatjuk.



1.3. ábra

1.4.3. A relációs adatmodell

A *relációs adatmodell* más filozófiára épül, mint az előbbi kettő. Ennél az adatmodellnél a három adatmodell-elem közül a kapcsolat nem játszik szerepet, pontosabban szólva a kapcsolat nem épül be az adatmodellbe. A relációs adatmodellnél a tulajdonságok kapják a fő hangsúlyt, a tulajdonságokkal definiáljuk az adatmodell szerkezetét. (Az olvasó bizonyára észrevette, hogy az előbbi két modellben a gráffal történő leírásban éppen a tulajdonságok nem szerepeltek, ezeket külön adtuk meg.)

A relációs adatmodellben az egyedet egy táblázattal adjuk meg. A táblázat oszlopai a tulajdonságok, sorai az egyed értékei (vagyis a táblázat maga éppen az egyedhalmaz). A táblázat egy-egy sorát a tulajdonságok konkrét értékei adják. Például a DOLGOZÓ egyed, pontosabban egyedhalmaz, amelynek tulajdonságai: név, születési-dátum, születési-hely, fizetés a következő táblázattal adható meg:

DOLGOZÓ

NÉV	SZÜLETÉSI DÁTUM	SZÜLETÉSI HELY	FIZETÉS
Kovács Pál	02/07/65	Pécs	15.000
Kis Péter	05/12/70	Budapest	13.000
Varga János	11/10/71	Sarkad	10.000
⋮	⋮	⋮	⋮
Tóth Edit	03/09/68	Pécel	9.600

A relációs adatmodellben létrehozott adatbázisokat több táblázattal adjuk meg (ne felejtjük el, minden tábla egy egyedhalmaz), de a táblázatok (tehát az egyedek) közötti kapcsolatokat nem definiáljuk az adatmodell felírásakor. Ez nem azt jelenti, hogy nincsen közöttük kapcsolat, de ezeket a kapcsolatokat például egyszerűen az fejezi ki, hogy két táblának van közös nevű (és természetesen közös tartalmú) oszlopa (mindkét táblában szerepel ugyanaz a tulajdonság).

A következő fejezetekben részletesebben ismertetjük a relációs és a hálós adatbázisokat, és röviden bemutatjuk a relációs SQL és a hálós IDMS adatbáziskezelő nyelvet.

2. FEJEZET

A RELÁCIÓS ADATMODELL

2.1. A relációs adatmodell meghatározása

Legyenek adottak az alábbi halmazok:

Vezetéknév	Keresztnév	Dátum	Település
Aba	Abigél	01/01/00	Battonya
⋮	⋮	⋮	⋮
Kovács	Károly	03/01/27	Budapest
⋮	⋮	⋮	⋮
Nagy	Péter	22/12/87	Pécs
⋮	⋮	⋮	⋮
Varga	Tamás	18/01/88	Sarkad
⋮	⋮	⋮	⋮

Vegyünk minden halmazból egy-egy elemet, és írjuk ezeket egymás mellé mindig ugyanabban a sorrendben. Például:

Aba	Tamás	01/01/00	Sarkad
Varga	Péter	03/01/27	Pécs
Varga	Károly	18/01/88	Battonya

Ha az összes vezetéknévből, keresztnévből stb. éppen azokat választjuk ki, amelyek „négyese” (négy elemből álló csoportja) pl. egy üzem összes dolgozójának vagy egy hallgatói évfolyamnak a fenti négy adattal való felsorolását jelenti, akkor azt mondhatjuk, hogy *relációs szerkezetű* adatbázist vagy röviden *relációs adatbázist* hoztunk létre. Ha a fenti adatok fölé oszlopneveket, illetve eléjük sorszámokat írunk, akkor a

	Vezetéknév	Keresztnév	Dátum	Település
1	Aba	Tamás	01/01/00	Sarkad
2	Varga	Péter	03/01/27	Pécs
3	Varga	Károly	18/01/88	Battonya

táblázathoz jutunk. A relációs adatbázis tehát egy *táblázat*.

A relációs elnevezést az indokolja, hogy a tábla sorait úgy képeztük, ahogy egy reláció elemeit képezzük a halmazelméleti *reláció* műveletben. Mint ismeretes, ezt a következőképpen definiáltuk.

Definíció: Legyenek $D_1, D_2, \dots, D_n$ adott halmazok. E halmazok *Descartes* (vagy *direkt*) *szorzatán* azt a D halmazt értjük, amelynek elemei az összes $(v_1, v_2, \dots, v_n)$ „érték n -esek”, ahol $v_1 \in D_1, v_2 \in D_2, \dots, v_n \in D_n$. Azaz veszünk egy elemet az első halmazból, a másodikból, a harmadikból és így tovább, és ebben a sorrendben egymás mellé írjuk őket, így megkapjuk a szorzathalmaz egy elemét. Ha ezt minden lehetséges módon meg tesszük, akkor létrehozuk az adott halmazok direkt szorzatát. A Descartes-szorzat egy R részhalmazát *relációnak* nevezzük. A $D_1, D_2, \dots, D_n$ halmazokat a reláció *tartományainak* (domain), a reláció egy elemét *n -esnek* (elem n -esnek, angolul *tuple*-nak) nevezzük. Az R reláció elemeit gyakran *rekordoknak* hívjuk.

Az előbbieken a relációs adatbázist a definíciónak megfelelő módon hoztuk létre, tehát a Vezetéknév, Keresztnév, Dátum, Település tartományokra támaszkodó relációt képeztünk.

Kis adatbázisunkban a táblázat utolsó két oszlopára is Dátum, ill. Település elnevezést írtunk, hiszen az oszlopban tényleg dátumok, települések szerepelnek. Az adatbázis szempontjából azonban ezek születési dátumot és születési helyet jelentenek. Amíg a Dátum nevű tartományban szerepel a dátum, addig sok mindent jelenthet (pl. a II. világháború végét), de ha egy személyi nyilvántartásba vesszük bele, akkor egy adott oszlopba írva nyilván születési dátum jelentése van (vagy egy másik oszlopba írva a munkába lépés dátuma). Ezt a különbséget úgy fejezzük ki, hogy az oszlopnak születési-dátum, születési-hely elnevezést adjuk, és azt mondjuk, hogy a tartományból az adatbázisnak egy *attribútumát* képeztük. A továbbiakban az adatbázis oszlopait attribútumoknak nevezzük. Úgy is fogalmazhatunk, hogy az attribútum egy változó, mely az oszlopba írt értékeket „veszi fel” a tartományban szereplő értékek közül, azaz a tartományt az adott attribútum értékészletének tekinthetjük.

Jelöljük a továbbiakban az attribútumokat az $A_1, A_2, \dots, A_n$ betűkkel, ezeket együtt (vagyis az attribútumok halmazát) A -val, azaz: $A = \{A_1, A_2, \dots, A_n\}$.

Az R relációs adatbázist legtöbbször táblázatnak fogjuk fel. Az attribútumok (az oszlopok) számát az R reláció *fokának*, a sorok számát pedig a reláció *számosságának* nevezzük. Ahelyett, hogy n -ed fokú, gyakran mondjuk, hogy n *változós*.

Ha a relációs adatbázist csak röviden akarjuk jelölni, akkor ezt

$$R\{A_1, A_2, \dots, A_n\}$$

alakban tesszük, ahol R a reláció neve, A_i pedig egy attribútum. A mi kis adatbázisunkat ennek megfelelően

NYILVÁNTARTÁS{VEZETÉKNÉV, KERESZTNÉV, SZÜLETÉSI-DÁTUM, SZÜLETÉSI-HELY}
alakban írjuk fel. Ez egy négy attribútumból álló adatbázis.

Az $R\{A_1, A_2, \dots, A_n\}$ alakot a *reláció sémájának* nevezzük. Ha ezt a sémát „kitöltjük”, azaz elkészítünk egy olyan táblázatot, amelynek oszlopai az $A_1, A_2, \dots, A_n$ attribútumok, és a táblázat soraiba konkrét értékeket írunk, akkor a relációs séma egy *értékét* kapjuk.

A NYILVÁNTARTÁS{VEZNÉV, KERNÉV, SZÜLDÁT, SZÜLHELY} relációs sémának egy értéke a

VEZNÉV	KERNÉV	SZÜLDÁT	SZÜLHELY
Kovács	Pál	03/04/72	Szeged
Nagy	István	07/06/68	Pécs
Bán	Péter	07/05/60	Monor
Szél	József	03/09/65	Pécel
Kiss	Lajos	16/05/62	Budapest

táblázat. (Az attribútumok neveit rövidítve írtuk.)

Vegyük az A attribútumhalmaznak egy B részhalmazát, mondjuk az A_1, A_3, A_5 attribútumokat, vagyis a $B = \{A_1, A_3, A_5\}$ halmazt. A reláció valamely sorának azokat az elemeit, amelyek a B halmazban szereplő oszlopokban (példánkban az A_1, A_3, A_5 oszlopokban) vannak, a *sor B -n felvett értékeinek* vagy egyszerűen *B értékeinek* nevezzük az adott sorban. A t sor értékeit a B halmazon $t(B)$ -vel jelöljük.

2.2. A relációs adatbázis belső szerkezete

A relációs adatmodell voltaképpen egy *adatszerkezet*. A szerkezet mélyebb elemzése az adatbázisban szereplő adatok közötti fontos kapcsolatokra, összefüggésekre mutathat rá. Kiderülhet például, hogy az attribútumok nem mind „egyenlőek” az adatbázis szempontjából; bizonyos attribútumok (vagy ezek egy csoportja együtt) kitüntetett szerepet kapnak.

2.2.1. A reláció kulcsai

Definíció: Az A attribútumhalmaz egy K részhalmazát *kulcsnak* nevezzük, ha 1. a K értékei az R reláció mindegyik sorát egyértelműen meghatározzák, 2. de ha egyetlen attribútumot is elhagyunk K -ból, akkor ez már nem teljesül.

Megjegyzés: egyes szerzők az ezen definícióval megadott K részhalmazt *minimális kulcsnak*, míg egy tetszőleges $K' \supset K$ halmazt, amelyik szintén egyértelműen meghatároz minden sort, *superkulcsnak* nevezik.

Ha K egyetlen attribútumból áll, akkor a kulcs *egyszerű*, ha nem ilyen, akkor *összetett*.

Nyilvánvaló, hogy egy relációban mindig van kulcs (esetleg a teljes attribútumhalmaz). Az is világos, hogy egy relációnak több kulcsa is lehet. A reláció attribútumai közül azokat, amelyek legalább egy kulcsban szerepelnek, *elsődleges* attribútumoknak, a többieket *másodlagosaknak* (nem elsődlegeseknek) nevezzük.

Példa: Legyen az adatbázis sémája a következő:

NYILVÁNTART{Vezetéknév, Keresztnév, Születési-dátum, Születési-hely,
Lakáscím, Anyja-neve, Személyszám}

Nyilvánvaló, hogy ebben a sémában a Személyszám attribútum kulcs (mégpedig egyszerű), mert egyértelműen meghatározza az adatbázis mindegyik elemét.

logikailag következik F -ből, vagy más szavakkal F logikailag implikálja $X \rightarrow Y$ -t, ha az R sémának minden olyan értékére (tehát egy konkrét táblázatra), amelyre teljesülnek az F -ben szereplő funkcionális függőségek, teljesül az $X \rightarrow Y$ függőség is.

Azt, hogy $X \rightarrow Y$ logikailag következik F -ből, így jelöljük: $F \models X \rightarrow Y$. Az előbb elmondott tranzitivitás tulajdonságot éppen ilyen logikai implikációval adtuk meg, azaz ezt így is írhatjuk:

$$\{P \rightarrow Q, Q \rightarrow S\} \models P \rightarrow S$$

Az F függőségi családból tehát új függőségek, azaz új függőségi család generálható. Ezt a családot F^+ -szal jelöljük, és az F lezárásának nevezzük, azaz

$$F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$$

Példa: Legyen $R\{B, C, D\}$ relációs séma (a séma tehát három attribútumból, B, C, D -ből áll), és legyen ebben $F = \{B \rightarrow C, C \rightarrow D\}$. Könnyen belátható, hogy az F^+ olyan $X \rightarrow Y$ funkcionális függőségekből áll, amelyek az alábbiakból kerülnek ki.

1. Az X a B -t tartalmazza, azaz

$$\{B, C, D\} \rightarrow \{B, C\}, \{B, C\} \rightarrow \{C, D\}, \{B\} \rightarrow \{D\}$$

2. Az X tartalmazza C -t, de nem tartalmazza B -t, azaz

$$\{CD\} \rightarrow C, C \rightarrow D, C \rightarrow \emptyset$$

(ahol $\emptyset$ az üres halmaz).

3. Az $X \rightarrow Y$ lehet $D \rightarrow D, D \rightarrow \emptyset$

A funkcionális függőségek tulajdonságai

Könnyen beláthatók a funkcionális függőségekkel kapcsolatban az alábbi tulajdonságok (szabályok).

1. *Reflexivitás*

Ha $Q \subseteq P \subseteq A$, akkor $P \rightarrow Q$ teljesül, azaz egy attribútumhalmaz a benne levő részhalmazt meghatározza. (Hiszen ha P -n megegyezik két sor, akkor a Q -n is, mert a Q attribútumok benne vannak P -ben is.)

2. *Bővítés*

Ha $P \rightarrow Q$ teljesül és $S \subseteq A$ egy tetszőleges attribútumhalmaz (A -ból), akkor

$$P \cup S \rightarrow Q \cup S$$

3. *Tranzitivitás*

Ha $P \rightarrow Q, Q \rightarrow S$, akkor $P \rightarrow S$ is teljesül.

4. *Egyesítési szabály*

$$\{P \rightarrow Q, P \rightarrow S\} \models P \rightarrow Q \cup S$$

5. *Pszudotranzitivitási szabály*

$$\{P \rightarrow Q, T \cup Q \rightarrow S\} \models P \cup T \rightarrow S$$

6. Dekompozíciós szabály

Ha $P \rightarrow Q$ teljesül és $S \subseteq Q$, akkor $P \rightarrow S$ is teljesül.

A fenti ún. Armstrong-axiómák (szabályok) alkalmazásával megadott funkcionális függőségekből újak származtathatók.

Érdekes és gyakorlatias kérdés: el lehet-e dönteni, hogy egy valamilyen $X \rightarrow Y$ funkcionális függőség logikailag kikövetkeztethető-e egy F függőségi családból az Armstrong-axiómák alkalmazásával.

Erre választ egy attribútumhalmaz *lezártjának* fogalmával tudunk adni, amit a következőképpen definiálunk.

Legyen F egy függőségi család az $R(A)$ reláción, és legyen X részhalmaza A -nak. Az X^+ *lezártja* X -nek F -re nézve az a Q attribútumhalmaz, amelyre az $X \rightarrow Q$ függőség az Armstrong-axiómák segítségével F -ből kikövetkeztethető.

Az X^+ segítségével a fenti kérdésre az alábbi választ fogalmazhatjuk meg: az $X \rightarrow Y$ függőség akkor és csak akkor következik egy F függőségi családból az Armstrong-axiómák alapján, ha az Y részhalmaza X^+ -nak, azaz $Y \subseteq X^+$.

Az X^+ az alábbi algoritmussal határozható meg.

1. Adott az $X \subseteq A$ és az F függőségi család.
2. Hozzuk létre az $X^{(0)}, X^{(1)}, \dots, X^{(n)}$ halmazokat a következő módon:
 - a) $X^{(0)} = X$
 - b) Az $X^{(i+1)}$ -et az $X^{(i)}$ -ből származtatjuk a következőképpen:
 - b_1) Vesszük az $X^{(i)}$ halmaz összes részhalmazát.
 - b_2) Megkeressük a függőségek között azokat (F -ből), amelyeknek bal oldalán szerepelnek ezek a részhalmazok.
 - b_3) Vesszük a b_1) pontban megtalált függőségek jobb oldalán szereplő halmazok unióját, legyen ez Q .
 - b_4) $X^{(i+1)} = X^{(i)} \cup Q$.
 - c) Nyilvánvaló, hogy $X = X^{(0)} \subseteq X^{(1)} \subseteq \dots X^{(i)} \subseteq \dots \subseteq A$. De mivel A véges halmaz, ezért egyszer az $X^{(i)} = X^{(i+1)}$ helyzet áll elő. Ez az $X^{(i)}$ lesz X^+ .

Példa: Legyen adva $R\{A, B, C, D, E, G\}$, és legyenek az F halmaz elemei (vagyis az R relációban fennálló funkcionális függőségek) a következők:

$\{A, B\} \rightarrow \{C\}$	$\{D\} \rightarrow \{E, G\}$
$\{C\} \rightarrow \{A\}$	$\{B, E\} \rightarrow \{C\}$
$\{B, C\} \rightarrow \{D\}$	$\{C, G\} \rightarrow \{B, D\}$
$\{A, C, D\} \rightarrow \{B\}$	$\{C, E\} \rightarrow \{A, G\}$

Legyen $X = \{B, D\}$. Hajtsuk végre az előbbi algoritmus lépéseit:

1. $X^{(0)} = \{B, D\}$
2. $X^{(1)}$ meghatározásához ki kell választani az F halmazból (azaz a fenti függőségekből) azokat, amelyeknek bal oldalán szerepel $\{B\}$, $\{D\}$ vagy $\{B, D\}$ (tehát a $\{B, D\}$ halmaz valamelyik részhalmaza). Ilyen egy van, a $D \rightarrow \{E, G\}$, tehát az E, G attribútumokat hozzávesszük az $X^{(0)}$ halmazhoz, és így megkapjuk $X^{(1)}$ -et: $X^{(1)} = \{B, D, E, G\}$.

3. Az R_3 halmazra (vagyis a ciklus harmadik lépésében) a következő helyzet áll elő:

$$Z = \{D\}((\{D\} \cap \{C, D\})^+ \cap \{C, D\}) = \{D\} \cup (\{D\}^+ \cap \{C, D\}) =$$

$$= \{D\} \cup (\{A, B, C, D\} \cap \{C, D\}) = \{C, D\}$$
Ebben a lépésben tehát Z megváltozott, a $\{C, D\}$ értéket vette fel ($Z = \{D\}$ -ből $\{C, D\}$ lett).
Az, hogy $\{D\}^+ = \{A, B, C, D\}$, abból következik, hogy az F^+ -ban mindegyik attribútum meghatározza a másikat.
4. Most az előbb kapott $Z = \{C, D\}$ -t véve és a ciklust végrehajtva, R_2 -re kapunk új Z -t, mégpedig $Z = \{B, C, D\}$ -t.
5. Végül $Z = \{B, C, D\}$ -re a ciklus az R_1 -re ad az előbbtől különböző Z -t, nevezetesen a $Z = \{A, B, C, D\}$ halmazt.
6. Ezzel az eljárás végére jutottunk, mert $Z = \{A, B, C, D\}$ -ben már nem lehet változás.

Azt kaptuk tehát, hogy $D^+ = \{A, B, C, D\}$, ami viszont tartalmazza A -t, a $D \rightarrow A$ függés jobb oldalát, vagyis: $G \models D \rightarrow A$. Mivel azt már beláttuk, hogy az F többi függősége benne van G^+ -ban (voltaképpen G -ben), tehát G és F ekvivalensek, ami azt jelenti, hogy a szétbontás megtartja a függőségeket.

Néhány speciális funkcionális függőség

Teljes függőség. Legyen $P, Q \subseteq A$ és $P \rightarrow Q$. Azt mondjuk, hogy Q teljesen függ (funkcionálisan) P -től az R relációban, ha Q a P egyetlen valódi részalmazától sem függ (funkcionálisan); azaz bárhogyan is képezünk a P attribútumaiból attribútumcsoportokat, ezek funkcionálisan nem határozzák meg Q -t. Ellenkező esetben *részleges függőségről* beszélünk.

Példa: A MERKUR{Név, Típus, Szín, Ár, Rendszám} relációs sémában a
 $\{\text{Név, Típus}\} \rightarrow \{\text{Ár}\}$

függőség teljes, mert az Ár nem függ (funkcionálisan) külön sem a Név-től, sem a Típus-tól. A Lada lehet 1200-as, 1300-as, 1500-as, amiknek az ára különböző, tehát abból, hogy a Név oszlopban két sorban Lada van, nem következik, hogy az Ár oszlopban egyenlő számok vannak. Ugyanígy ha a Típus oszlop két sorában Combi szerepel, ez nem jelenti azt, hogy az Ár ugyanaz (hiszen Combi lehet Trabant is, meg Dacia is).

Tranzitív függőség. Legyen $P, S \subseteq A$ és $P \rightarrow S$. Azt mondjuk, hogy S tranzitíven függ P -től, ha létezik olyan $Q \subseteq A$, hogy $P \rightarrow Q$, $Q \rightarrow S$, de úgy, hogy $Q \rightarrow P$, $S \rightarrow Q$ nem teljesül.

Példa: Tekintsük a következő relációt:

NYILVT{NÉV, SZEMSZÁM, HAVI-BÉR, ÉVES-BÉR}

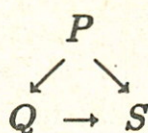
Legyen az ÉVES-BÉR a HAVI-BÉR 12-szerese. A kulcs a SZEMSZÁM (személyi szám). Nyilvánvaló, hogy

$$\{\text{SZEMSZÁM}\} \rightarrow \{\text{HAVI-BÉR}\}$$

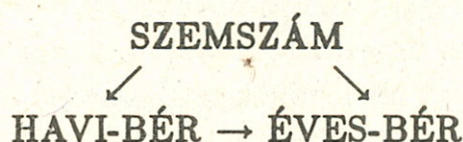
$$\{\text{SZEMSZÁM}\} \rightarrow \{\text{ÉVES-BÉR}\}$$

Ugyanakkor $\{\text{HAVI-BÉR}\} \rightarrow \{\text{ÉVES-BÉR}\}$ is teljesül. Tehát $P = \text{SZEMSZÁM}$, $S = \text{ÉVES-BÉR}$, $Q = \text{HAVI-BÉR}$ jelöléssel $P \rightarrow S$ teljesül, és létezik olyan Q , amelyre $P \rightarrow Q$, $Q \rightarrow S$ érvényes.

A tranzitív függőség rajzban a következőképpen adható meg:



Példánkban



Generátor attribútumhalmaz. Legyen $P, Q \subseteq A$ és $P \rightarrow Q$. A P attribútumhalmazt a Q attribútumhalmaz *generátorának* hívjuk, ha a P elemeinek száma (az attribútumok száma) kisebb, mint Q elemeinek száma, és Q teljesen függ P -től.

Az előbbi példában a

$$\{\text{SZEMSZÁM}\} \rightarrow \{\text{NÉV}, \text{HAVI-BÉR}, \text{ÉVES-BÉR}\}$$

funkcionális függőségben a nyíl bal oldalán kevesebb attribútum van, mint a jobb oldalon, tehát a SZEMSZÁM generátor.

A fenti függőséghez hasonló másfajta funkcionális függőségek is definiálhatók egy relációs adatbázisban. Ezekre itt nem térünk ki, de felsoroljuk őket: *gyenge funkcionális függőség, erős funkcionális függőség, duális funkcionális függőség, többértékű függőség.*

2.2.3. Dekompozíció

Az adatbázison belül előforduló mély kapcsolatra példaként egy másik „strukturális” tulajdonságot ismertetünk – nagyon röviden – a következőkben. Legyen A valamilyen attribútumhalmaz, és legyenek X, Y ennek részhalmazai úgy, hogy uniójuk éppen A -t adja ki, azaz legyen $X \subseteq A, Y \subseteq A$ és $X \cup Y = A$.

Azt mondjuk, hogy X és Y *dekompozíciós* párt alkot az R relációban, ha R minden olyan két t_i, t_j sora esetén, amelyekre $t_i = t_j$ teljesül az $X \cap Y$ metszethalmazon (azaz t_i értéke megegyezik t_j értékével a metszethalmazon), következik hogy létezik olyan $t \in R$, amelyre $t = t_i$ az X -en, $t = t_j$ az Y -on. Ezt a meghatározást a következőképpen is felírhatjuk: $\forall t_i, t_j \in R$ esetén, melyre $t_i(X \cap Y) = t_j(X \cap Y), \exists t \in R$, melyre $t(X) = t_i(X)$ és $t(Y) = t_j(Y)$.

Ha az X, Y párra teljesülnek ezek a tulajdonságok, akkor ezt így jelöljük: (X, Y) .

Lássunk a definícióra egy példát. Legyen az adatbázis (részlet) a következő:

	X		Y		
	A_1	A_2	A_3	A_4	
t_1	d	a	f	e	$X = (A_1, A_2, A_3)$ $Y = (A_2, A_3, A_4)$ $X \cap Y = (A_2, A_3)$
t_2	c	a	f	b	
t_3	d	a	f	b	
t_4	.	.	.	.	

Jól láthatjuk, hogy X és Y dekompozíciót alkotnak, hiszen:

- t_1 az $X \cap Y$ -on megegyezik t_2 -vel (az $\{A_2, A_3\}$ halmazon mindkettő a, f értékeket vesz fel);
- a t_3 az X -en ugyanazt az értéket veszi fel, mint t_1 (a d, a, f értékeket), az Y -on pedig ugyanazt, mint a t_2 (az a, f, b értékeket).

Előfordulhat, hogy egy R relációban nemcsak egyetlen (X, Y) pár alkot dekompozíciót. Az összes ilyen párt az R reláció *teljes dekompozíciós családjának* hívjuk, és D_R -rel jelöljük. Illusztrációként felsoroljuk a D_R néhány tulajdonságát.

- Az üres halmaz és az A attribútumhalmaz dekompozíciót alkot (azaz $(\emptyset, A) \in D_R$).
- Ha $X \subseteq A$, akkor (X, A) dekompozíció.
- A dekompozíció szimmetrikus: ha (X, Y) dekompozíciós pár, akkor (Y, X) is az.
- Ha (X, Y) dekompozíció és $Z \subseteq A$, akkor $(X \cup Z, Y \cup Z)$ is dekompozíció.
- Ha (X, Y) és (Z, W) is dekompozíció és $Z \cap Y = Y$, akkor $(X \cap Z, W)$ is dekompozíció.

A dekompozíció fogalma segítségével számos más fogalom definiálható, amelyek a relációs adatbázis „belső szerkezetéről” sok mindent elárulnak, érdekes tulajdonságokat fejeznek ki. Ezekkel itt nem foglalkozunk.

2.2.4. Normálformák

Az adatbázisok belső szerkezetét jellemzik az ún. *normálformák* is, amikből az alábbiakban négyet sorolunk fel.

Definíció. Egy R relációról azt mondjuk, hogy *1. normálformájú* (1. normálformában van, 1NF típusú), ha minden sorában pontosan egy attribútumérték áll. A definíció jelentését az alábbi példa jól megvilágítja. Legyen az adatbázisunk a következő:

	Vezetéknév	Keresztnév	Szakképzettség	Születési dátum
t_1	Nagy	Zsolt	gépészmérnök közgazdász	16/02/52
t_2	Kiss	Pál	lakatos	08/08/58

Mint látjuk, a t_1 sorban a szakképzettség attribútumhoz két „érték” (gépészmérnök, közgazdász) tartozik. Ez azt jelenti, hogy egy valamilyen „szakképzettségek” nevű domainből (ami mondjuk tartalmazza a Magyarországon előforduló összes szakképzettséget) ebbe a sorba két értéket vettünk. (Azt szoktuk mondani ilyenkor, hogy az attribútumérték *nem elemi*). Ez az adatbázis tehát nincsen 1. normálformában.

A nem 1NF típusú adatbázis könnyen ilyené alakítható: a több attribútumértéket tartalmazó sort annyi sorra bontjuk, amennyi benne az attribútumértékek száma.

Példánkat az alábbi alakban írjuk fel:

	Vezetéknév	Keresztnév	Szakképzettség	Születési dátum
t_1	Nagy	Zsolt	gépészmérnök	16/02/52
t_2	Nagy	Zsolt	közgazdász	16/02/52
t_3	Kiss	Pál	lakatos	08/08/58

Definíció. Egy R relációt *2. normálformájúnak* (2NF típusúnak) nevezünk, ha 1. normálformájú, és ha minden másodlagos attribútum a reláció bármely kulcsát teljesen függ. (Példaként lásd alább az AUTÓSBOLT relációt.)

Vegyük észre az alábbiakat:

1. Ha minden kulcs egyetlen attribútumból áll, akkor ha a reláció 1NF-ben van, egyúttal 2NF típusú is.
2. Ha a relációban nincsen másodlagos attribútum, akkor minden 1NF típusú reláció 2NF típusú is.

A nem 2NF típusú relációt könnyen átalakíthatjuk ilyenekké. A módszert mindjárt bemutatjuk egy példán. Előtte azonban nézzük a harmadik és negyedik típust.

Definíció. A 2. normálformájú relációt *3. normálformájúnak* (3NF típusúnak) nevezzük, ha egyetlen másodlagos attribútum sem függ tranzitíven valamely kulcstól.

Ha a reláció nincsen 3NF-ben, akkor szét tudjuk bontani olyanokká, amelyek már 3NF típusúak (lásd alább az AUTÓSBOLT relációt). Az AUTÓSBOLT reláció nincsen 3NF-ben, mert van benne tranzitív függőség.

Definíció. Az 1. normálformájú relációról azt mondjuk, hogy *Boyce-Codd normálformájú* (BCNF-típusú), ha a reláció minden generátora egyben kulcs is.

Az adatbázis normalizálása

Azt az eljárást, melynek folyamán egy adatbázist valamilyen normálformára hozunk, *normalizálásnak* nevezzük. A 2NF-re, 3NF-re való hozást úgy végezzük el, hogy az eredetileg nem ilyen formában lévő adatbázisból több olyan adatbázist készítünk, amelyek már a kívánt normálformában vannak.

Lássunk egy példát a normálformák lényegének megvilágítására, és arra, hogyan lehet 2NF-re, 3NF-re hozni egy relációt. Tegyük fel, hogy egy autósbolt naponta az alábbi adatokat gyűjti — példaként ki is töltjük a formanyomtatványt.

AUTÓSBOLT

Dátum	Alkatrésznév	Alkatrészkód	Egységár	Darab
02/03/88	kuplung	TX-5	1500	2
	elosztófej	XB-3	150	6
	kondenzátor	ET-2	30	4
	fékfőhenger	F-6	120	1

Összérték: 4140

Befizetés: 3140

A Darab oszlopban azt írjuk fel, hogy az adott alkatrészből azon a napon hányat adtak el. Az Alkatrészkód ugyanazon alkatrésznél kocsitípusonként különbözik egymástól.

Tegyük fel, hogy amennyiben a napi bevétel (összérték) az 1000 Ft-ot meghaladja, úgy a különbséget a postán befizetik (1000 Ft készpénzt meghagynak).

Abból a célból, hogy ne kelljen hosszú neveket írunk az adatbázisban, vezessük be a következő rövidítéseket: Dátum: DÁT; Alkatrésznév: ANÉV; Alkatrészkód: AKÓD; Egységár: EÁR; Darab: DB; Összérték: ÖÉRT, Befizetés: BEFIZ.

A napi adatokból az alábbi relációs sémát állítjuk össze:

AUTÓSBOLT (DÁT, ANÉV, AKÓD, EÁR, DB, ÖÉRT, BEFIZ)

Írjuk fel az adatbázist mindjárt 1NF-ben (vagyis a dátumot, az összértéket és a befizetést annyiszor írjuk le új sorban, ahány „tétel” azon a napon van).

AUTÓSBOLT

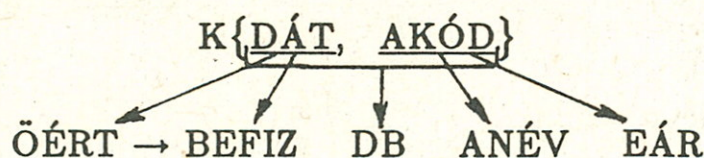
DÁT	ANÉV	AKÓD	EÁR	DB	ÖÉRT	BEFIZ
02/03/88	KUPLUNG	TX-5	1500	2	4140	3140
02/03/88	ELOSZTÓFEJ	XB-3	150	6	4140	3140
02/03/88	KONDENZÁTOR	ET-2	30	4	4140	3140
02/03/88	FÉKHENGER	F-6	120	1	4140	3140
02/04/88	FÉKHENGER	F-6	120	1	4620	3620
02/04/88	KUPLUNG	TX-5	1500	3	4620	3620
02/05/88	ELOSZTÓFEJ	XB-3	150	2	3600	2600
02/05/88	VÍZPUMPA	P-12	1100	3	3600	2600

Nyilvánvaló, hogy a $K = \{DÁT, AKÓD\}$ halmaz kulcs, hiszen értékeik egyértelműen meghatároznak minden sort, vagy ami ugyanaz,

$\{DÁT, AKÓD\} \rightarrow \{ANÉV, EÁR, DB, ÖÉRT, BEFIZ\}$

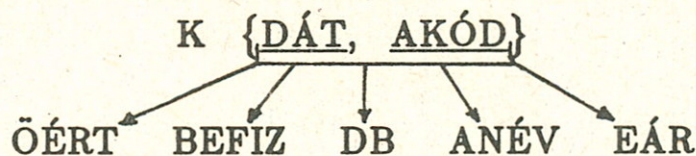
teljesül, hiszen ha két sorban a dátum és az alkatrészkód megegyezik (vagy megegyezne), akkor a többi értéknek is meg kell (kellene) egyeznie. Adatbázisunkban ugyan ténylegesen nincsen olyan két sor, amelyben a dátum és az alkatrészkód megegyezik, de a szerkezet azt „diktálja”, hogy amennyiben volna ilyen, akkor meg kellene egyeznie ebben a két sorban a többi attribútumértékeknek is. (A táblázat alapos elemzése bizonyára meggyőzi erről az olvasót.)

Adatbázisunkban az alábbi funkcionális függőségek vannak:



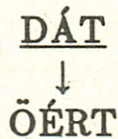
Az ábrán $K\{ \}$ -vel jelöltük a kulcsot, a kulcstól, illetve annak részeitől a funkcionális függőségeket pedig jelölje $\rightarrow$.

Azt, hogy a

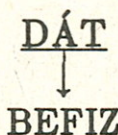


függőségek léteznek, éppen az előbb láttuk be (hiszen a kulcs funkcionálisan meghatározza a másodlagos attribútumokat).

Könnyen beláthatjuk azt is, hogy a

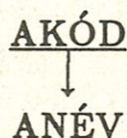


függőség is fennáll. Ugyanis nyilvánvaló, hogy ha két dátum megegyezik, akkor abban a két sorban az ÖÉRT értékei is megegyeznek, hiszen egy dátummal egy sorban az adott napon beszedett bevétel szerepel. A



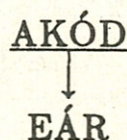
funkcionális függőség fennállása az előbbi módon látható be.

Az



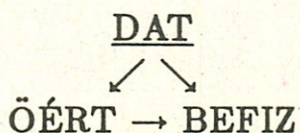
függőség azt fejezi ki, hogy ha valamely két sorban az AKÓD értéke megegyezik, akkor ott az ANÉV oszlopban lévő értékeknek is meg kell egyeznie. Ez valóban teljesül, hiszen egy adott alkatrészkóddal jelölt alkatrész neve mindig ugyanaz.

Hasonló a helyzet az

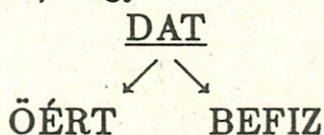


függőséggel is.

A



hármas egy tranzitív függőség. Azt, hogy a



teljesül, az előbb beláttuk. Csak azt kell megvizsgálni, fennáll-e az ÖÉRT → BEFIZ függőség. Ez nyilvánvaló, mert a BEFIZ oszlop értékei minden sorban 1000-rel kisebbek, mint az ÖÉRT oszlop elemei, tehát ha két sorban az ÖÉRT ugyanaz, akkor a BEFIZ-ben is megegyeznek az értékek.

Megmutattuk tehát, hogy a *másodlagos attribútumok nem csak a kulcstól függenek, hanem annak részhalmazaitól is* (a kulcs két attribútumától külön-külön). Ez azt jelenti, hogy adatbázisunk nincsen 2NF-ben. De ha 2NF-ben volna is, a tranzitív függőség miatt 3NF akkor sem teljesül.

Bontsuk fel adatbázisunkat olyan adatbázisokra, amelyek együtt megtartják a fenti funkcionális függőségeket (azaz a felbontást úgy végezzük el, hogy ne veszítsünk el egyetlen függőséget sem a tranzitív függőség kivételével), ugyanakkor legyenek 2NF-ben (ill. 3NF-ben).

A fenti feltételeknek eleget tevő felbontás algoritmus egyszerű: kiemeljük az eredeti adatbázisból azokat a részeket, amelyek a „bajt” okozzák (tehát ahol a másodlagos attribútumok függenek a kulcs részeitől is), és belőlük új relációkat hozunk létre. Az eljárás lényegét a 2.1. ábrán láthatjuk (bekereteztük a szétválasztandó részeket).

Készítsük el az AUTÓSBOLT relációból az alábbiakat:

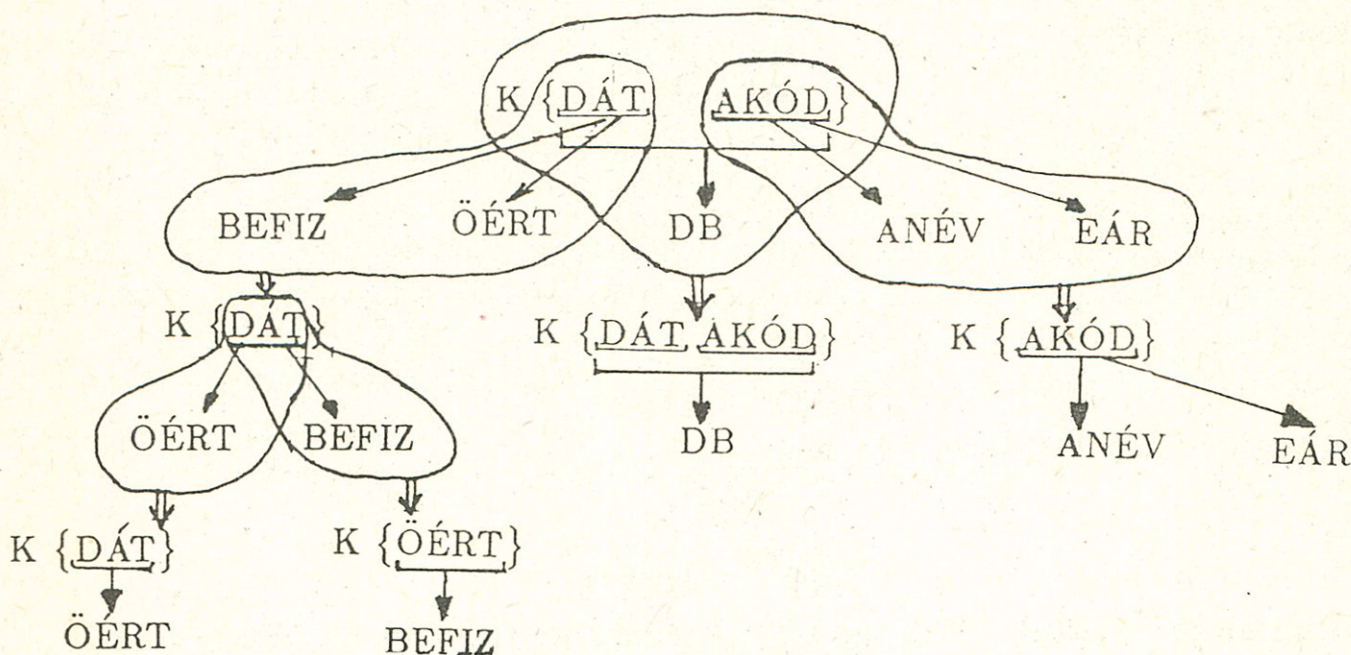
a) NAPIELAD (DÁT, ÖÉRT, BEFIZ)

DÁT	ÖÉRT	BEFIZ
02/03/88	4140	3140
02/04/88	4620	3620
02/05/88	3600	2600

A funkcionális függőségek:

$K \{ \underline{DÁT} \}$
 $\swarrow \quad \searrow$
 $\underline{ÖÉRT} \rightarrow \underline{BEFIZ}$

Mint látjuk, ez az adatbázis 2NF-ben van, hiszen a K kulcs egyetlen attribútumból áll. Nincsen azonban az adatbázis 3NF-ben, mivel a tranzitív függőség nem szűnt meg. „Bontsuk fel” ezt az adatbázist két adatbázisra.



2.1. ábra

a1) NAPIELAD1 (DÁT, ÖÉRT)

DÁT	ÖÉRT
02/03/88	4140
02/04/88	4620
02/05/88	3600

A funkcionális függőségek:

$K\{\underline{DÁT}\}$
 $\downarrow$
 ÖÉRT

Ez az adatbázis 2NF-ben van (egyetlen attribútumból áll a kulcs) és nincsen benne tranzitív függőség, tehát 3NF-ben is van.

a2) NAPIELAD2 (ÖÉRT, BEFIZ)

ÖÉRT	BEFIZ
4140	3140
4620	3620
3600	2600

A funkcionális függőségek:

$K\{\underline{ÖÉRT}\}$
 $\downarrow$
 BEFIZ

Ez az adatbázis is 2NF-ben van (egyetlen attribútumból áll a kulcs), és mivel tranzitív függőség sincsen benne, ezért 3NF-ben is van.

Az eredeti adatbázisból elkészített NAPIELAD1 és NAPIELAD2 adatbázisokkal rendbe tettük azt a részt, amely azzal okozott gondot a normálformák tekintetében, hogy a BEFIZ, ÖÉRT másodlagos attribútumok nemcsak a $K\{DÁT, AKÓD\}$ kulcstól függtek, hanem annak egy részhalmazától (a DÁT attribútumtól) is, és ráadásul tranzitív függőség is szerepelt benne.

b) ALKATRÉSZ (AKÓD, ANÉV, EÁR)

AKÓD	ANÉV	EÁR
TX-5	KUPLUNG	1500
XB-2	ELOSZTÓFEJ	150
ET-2	KONDENZÁTOR	60
F-6	FÉKHENGER	120
P-12	VÍZPUMPA	1100

A funkcionális függőségek:

$K\{\underline{AKÓD}\}$
 $\swarrow \searrow$
 ANÉV EÁR

Mivel a másodlagos attribútumok olyan kulcstól függnek, amely egyetlen attribútumból áll, ezért az adatbázis 2NF-ben van (és 3NF-ben is!).

c) ELADÁS (DÁT, AKÓD, DB)

DÁT	AKÓD	DB
02/03/88	TX-5	2
02/03/88	XB-3	6
02/03/88	ET-2	4
02/03/88	F-6	1
02/04/88	F-6	1
02/04/88	TX-5	3
02/05/88	XB-3	2
02/05/88	P-12	3

A funkcionális függőségek:

$K\{\underline{DÁT}, \underline{AKÓD}\}$
 $\swarrow \searrow$
 DB

Ez az adatbázis is 2NF-ben van, mert a másodlagos attribútum nem függ a kulcsot alkotó attribútumok részhalmazától (külön egyiktől sem függ), és egyúttal 3NF-ben is van, mert nincsen benne tranzitív függőség.

Mint látjuk, az eredeti adatbázisból négy olyan adatbázist készítettünk, amelyek mindegyike 3NF-ben van.

Anomáliák az adatbázisban

Az egyes normálformák az adatbázisban meglévő bizonyos anomáliákat szüntetik meg. Ezek közül most azokkal foglalkozunk, amelyek a 2NF-re hozással szűrhetők ki.

Módosítási anomália. Előfordulhat, hogy valamely attribútum értékeit módosítani kell. Ha például az AUTÓSBOLT adatbázisban az alkatrész kódokon akarunk változtatni, akkor az AKÓD oszlop minden elemét meg kell vizsgálni, és minden olyan sorban végre kell hajtani a módosítást, amelyben a módosítandó kód szerepel. Az ALKATRÉSZ relációban viszont az AKÓD oszlopban minden kód csak egyszer szerepel, tehát csak egy helyen kell módosítani. Azt mondhatjuk tehát, hogy a 2NF-re hozással a módosítási anomáliát szüntettük meg.

Törlési anomália. Az adatbázisunk lehet olyan, hogy ha benne egy sort törölünk, alapvető információkat veszítünk el. Ha pl. az AUTÓSBOLT adatbázisban a 02/03/88; ET-2 kulcsú rekordot töröljük, akkor ez a kondenzátorra vonatkozó információk elvesztését eredményezné. Viszont ha a törlést az ELADÁS adatbázisban végezzük el, nincsen baj, mert az ALKATRÉSZ adatbázisban mindig megmarad ez az információ. A 2NF-es forma tehát megszünteti ezt az anomáliát.

Bővítési anomália. A bővítési anomália fogalmát is példánkon mutatjuk be. Tegyük fel, hogy az AUTÓSBOLT relációt egy új információval kellene bővíteni, mondjuk az ékszíj alkatrészszel, aminek kódja: EX-5, ára: 25 Ft. Ezeket az információkat az eladó ismeri, de nem tudja beírni az AUTÓSBOLT relációba, mert eddig még nem adtak el ilyen alkatrészt, és így nincsen kulcsunk rá. Az ALKATRÉSZ relációba viszont könnyen bevihetők az ékszíjról szóló adatok. Ez azt jelenti, hogy a 2NF-es relációban nincsen bővítési anomália.

2.3. A relációs algebra

A relációs adatmodell szerint felépített adatbázisokat (a relációs adatbázisokat) speciális *adatbáziskezelő nyelvekkel* dolgozzák fel. Az adatbáziskezelő nyelvek fontos eleme a *lekérdezési* lehetőség. Ebben a tekintetben a nyelvek egy közös alapra a *relációs algebrára* támaszkodnak, lekérdezési eljárásaikat a *predikátumkalkulusra* építik. A relációs algebra és a predikátumkalkulus lényegében absztrakt adatbáziskezelő nyelvek, voltaképpen arra valók, hogy velük leírjuk azt a minimumot, amit egy relációs adatbáziskezelő nyelvtől elvárunk. (Az adatbáziskezelő eljárások metanyelvének is lehet tekinteni őket.) Könyvünkben csak a relációs algebrát tárgyaljuk; azt is röviden, inkább csak illusztrálni akarjuk a fontosabb műveleteket.

A relációs algebra műveletei

1. Relációk uniója (a műveleti jel: $\cup$)

Legyen R és S két n -ed fokú reláció. Ezek unióján (egyesítésén) azt a $R \cup S$ szintén n -ed fokú relációt értjük, amelynek rekordjai vagy az R -nek, vagy az S -nek, vagy mindkettőnek elemei. A műveletet az alábbi példán szemléltetjük:

R	A B C		S	D E F		$R \cup S$
	a b c	$\cup$		b d e	$=$	a b c
	b d e			a d b		b d e
	f c b					f c b
						a d b

(A közös elemet, (b, d, e) -t egyszer szerepeltetjük.)

Az $R \cup S$ eredményreláció attribútumait nem neveztük meg, mert erre nézve az R és S relációk attribútumai nem adnak információt. Természetesen ha mindkét relációban ugyanazon attribútumok szerepelnek, akkor az $R \cup S$ attribútumai ezek lesznek.

2. Relációk különbsége (a műveleti jel: $-$)

Az n -ed fokú R és S relációk $R - S$ különbségén azt az n -ed fokú relációt értjük, amelynek elemei R -nek elemei, de S -nek nem.

Példa:

R	A B C		S	F E D		$R - S$
	a b c	$-$		f a d	$=$	a b c
	f a d			b c d		d b c
	d b c					

3. A Descartes-szorzat (a műveleti jel: $\times$)

Legyen R n_1 -ed fokú, S n_2 -ed fokú reláció. Ezek Descartes-szorzatán azt az $(n_1 + n_2)$ -ted fokú $R \times S$ relációt értjük, amelyben minden elem első részét az R relációból, második részét pedig az S relációból vesszük.

Példa:

R	A B C		S	E F D		$R \times S$	A B C E F D
	c b a	$\times$		a g b	$=$		c b a a g b
	f a d			f a d			c b a f a d
	d b c						f a d a g b
							f a d f a d
							d b c a g b
							d b c f a d

Az $R \times S$ reláció attribútumneveit itt meg tudjuk adni, ha különböznek egymástól. Azokat az attribútumneveket amelyek azonosak, a reláció nevével jelöljük meg. (Például ha R -ben, is S -ben is van A attribútum, akkor az R -beli neve $R.A$, az S -belié pedig $S.A$ lesz.)

4. A projekció művelete (a műveleti jel: π)

Projekció műveletet végzünk a reláción, ha kiemelünk belőle néhány megadott attribútumot (oszlopával együtt), és a maradékot adott szempont szerint átrendezzük. Az így kapott relációt az eredeti projekciójának nevezzük, és $\pi_{i_1, i_2, \dots, i_m}(R)$ -l jelöljük, ahol $i_1, i_2, \dots, i_m$ a meghagyott és átrendezett attribútumok nevei.

Példa:

R	A B C	$\pi(R)$ A, C	$\pi(R)$ B, A
	a b c	a c	b a
	f a d	f d	a f
	d b c	d c	b d

Megjegyzés: A π betű mellé gyakran nem az attribútumneveket, hanem az attribútumok által képviselt oszlopok sorszámát (balról jobbra vége) írjuk. Példánkban $\pi_{A,C}(R)$ helyett $\pi_{1,3}(R)$ -t írhatunk.

5. A szelekció művelete (jele: σ_F)

Az R reláción végzett szelekció egy olyan $\sigma_F(R)$ relációt eredményez, amelynek elemei az R reláció elemeiből kerülnek ki az F formula által meghatározott módon.

Az F formula az alábbi objektumokból áll:

1. Operandusok (konstansok, attribútumnevek vagy attribútum-sorszámok)
2. Az operandusokon értelmezett aritmetikai relációk ($<$, $>$, $=$, $\leq$, $\geq$, $\neq$)
3. Az operandusokon értelmezett logikai műveletek: $\wedge$ (és), $\vee$ (vagy), $\neg$ (nem). A művelet lényegét a példán jól láthatjuk.

Példa:

R	A B C	$\sigma(R)$ $A="a" \vee B="b"$
	a b c	a b c
	d e b	a d f
	f g a	f g a
	a d f	

Az R relációból azokat a sorokat választottuk ki, amelyekben az A oszlopban 'a', vagy a B oszlopban 'b' van.

6. Relációk hányadosa (a műveleti jel: $\div$)

Legyen R n -ed fokú, S m -ed fokú reláció ($n > m$). Az R és S relációk hányadosán azt az $R \div S$ relációt értjük, amely $n - m$ oszlopból áll, és amelynek Descartes-szorzata az S relációval benne van az R relációban; azaz

$$(R \div S) \times S \subseteq R$$

(A hányados tehát a Descartes-szorzat speciális „inverzének” tekinthető.) A művelet lényege jól látszik a példán.

Példa:

R	A B C D		S	E F		R ÷ S
	a b c d			c d		a b
	a b e f	÷		e f	=	
	b c e f					e d
	e d c d					
	e d e f					
	a b d e					

Szorozzuk meg a hányadost az „osztó”-val, az eredmény valóban része lesz az R relációnak.

7. Relációk metszete (jele: $\cap$)

Két n -ed fokú reláció, R és S $R \cap S$ -sel jelölt metszetén az $R - (R - S)$ relációt értjük.

Példa:

R	A B C		S	D A E		R ∩ S
	a a b			a a b		a a b
	a c b	∩		b c d	=	b c d
	b c d			e f g		
	a a d					

8. Az összekapcsolás művelete (jele: $\bowtie$)

Legyen R n -ed fokú, S pedig m -ed fokú reláció. Ezek összekapcsolásán egy olyan $(n + m)$ -ed fokú $R \bowtie S$ -sel jelölt relációt értünk, amelynek elemei az $R \times S$ elemei közül azok, amelyekre az R reláció i és az S reláció j attribútuma között teljesül a θ -val jelölt aritmetikai reláció ($<, >, =, \leq, \geq, \neq$). A szelekció műveletével a definíció a következőképpen írható fel:

$$R \bowtie_{i\theta j} S = \sigma_{i\theta(n+j)}(R \times S)$$

Példa:

R	A B C		S	D E		R ⋈ S
	1 2 3			3 1		A B C D E
	4 5 6	⋈		6 3	=	1 2 3 3 1
	7 8 9	B < D				1 2 3 6 2
						4 5 6 6 2

9. A természetes összekapcsolás (jele: $\bowtie$)

Legyen R n -ed fokú, S m -ed fokú reláció. A természetes összekapcsolás műveletével képzett $R \bowtie S$ -sel jelölt relációt a következőképpen kapjuk:

a) Képezzük $R \times S$ -t.

b) Vegyük az azonos nevű (jelöljük A -val) attribútumokat R -ből és S -ből, és emeljük ki azokat a rekordokat $R \times S$ -ből, amelyekben az $R.A$ és $S.A$ oszlopban lévő értékek (mindegyik azonos nevű oszlopra) megegyeznek. (Az $R.A$, $S.A$ értelmezését lásd a Descartes-szorzat definíciójánál.)

c) Minden, a b) pontban vett attribútumra emeljük ki az $S.A$ oszlopokat (azaz vegyük ezen oszlopok nélküli relációt).

Példa:

a) lépés:

R	A B C		S	B C D		$R \times S$	A R.B R.C S.B S.C S.D
	a b c			b c d			a b c b c d
	d b c	$\times$		b c e	=		a b c b c e
	b b f			a d b			a b c a d b
	c a d						d b c b c d
							d b c b c e
							d b c a d b
							b b f b c d
							b b f b c e
							b b f a d b
							c a d b c d
							c a d b c e
							c a d a d b

b) lépés: Kiválasztjuk az $R.B$ és $S.B$, valamint az $R.C$ és $S.C$ oszlopokban is megegyező sorokat:

A	R.B	R.C	S.B	S.C	S.D
a	b	c	b	c	d
a	b	c	b	c	e
d	b	c	b	c	d
d	b	c	b	c	e
c	a	d	a	d	b

(Azokat a rekordokat vettük ki, amelyekre $R.B = S.B$, $R.C = S.C$.)

c) lépés: Szüntessük meg a dupla oszlopokat, hagyjuk el az $S.B$, $S.C$ oszlopokat.

$R \bowtie S$	A B C D
	a b c d
	a b c e
	d b c d
	d b c e
	c a d b

2.4 Relációs adatbáziskezelő nyelvek

A relációs adatmodellre 1970-től kezdve számos adatbáziskezelő nyelvet dolgoztak ki. A korai próbálkozások közül egyetlen rendszer sem terjedt el igazán. Voltak olyan vélemények, még a hetvenes évek végén is, hogy a relációs adatmodellre nem is lehet „gazdaságos” kezelőrendszert írni. Nagy méretek esetén még az igen nagy számítógépek is kicsinek bizonyultak.

A terjedelem nem engedi meg, hogy részletesen szóljunk az ismertebb „nagygépes” relációs adatbáziskezelő nyelvekről, pedig érdekes volna elemezni, hogy mennyire valószínűtlenek még ezek a relációs algebra műveleteit. Egy rövid történeti áttekintés talán nem felesleges.

Az egyik ilyen rendszer az ISBL (Information System Base Language) volt. Ez a hetvenes éves közepén készült el (IBM, Anglia). Az ISBL-lel egyidőben írták meg a SQUARE-nyelv fordítóprogramját, majd erre alapozva dolgozták ki SEQUEL nyelvet (1974-ben szintén az IBM-nél), illetve SQL-t (1976-ban). Később létrehozták a QUEL lekérdező nyelvet (Kalifornia Egyetem, Berkeley), ami már UNIX operációs rendszer alatt működött. Megemlítenéd még a QBE, AWK, SIDA nyelvek (a 70-es évek második feléből). Ezek az adatbáziskezelő rendszerek azonban nem terjedtek el; sokan azt „jósolták” ebben az időben, hogy a relációs adatmodellnek „bealkonyult”.

Senki sem számított arra, hogy éppen a PC-k megjelenése hozza igazán divatba a relációs adatmodellt. 1982-ben az Ashton-Tate cég létrehozta a dBase II, majd 1984-ben a dBase III relációs adatbáziskezelő nyelvet. Majd jött a dBase III+, később a FoxBase, PARADOX és sok más PC-re írt relációs nyelv. 1982-ben megszületett nagygépekre az IBM-nél az SQL/DB rendszer, amelyet az első relációs, kereskedelmi forgalomba kerülő nagygépes adatbáziskezelő rendszernek tekintenek. 1983-ban az IBM piacra dobta az SQL-re alapozott DB2 nevű adatbáziskezelő rendszert, amelynek alapján az ANSI (az Amerikai Szabványügyi Hivatal) 1986-ban szabványosította az SQL (Structured Query Language = strukturált lekérdező nyelv) nyelvet.

A szabványosítás jelentős fordulatot hozott a relációs adatbáziskezelő nyelvek területén. Ugyan a dBase-t relációs nyelvnek tekintjük, de nem teljességgel valószínűtlenek meg benne a Codd által kimondott elveket, ill. a relációs algebra műveleteit.

Az SQL viszont maradéktalanul kielégíti ezeket. Az SQL-szabvány nem „igazi” önálló programozási nyelv (nincs benne ciklusutasítás stb.), hanem mint a neve is kifejezi: lekérdező nyelv (természetesen az adatbázis definiálására is alkalmas). Voltaképpen csak más nyelvekkel együtt (COBOL, PL-I, C) válik programozási eszközzé. (Az ilyen nyelveket „beépülő-típusú” nyelveknek nevezzük.)

Az SQL-re alapozva számos adatbáziskezelő rendszert létrehoztak. Ezek a rendszerek az SQL-ből úgy keletkeztek, hogy „valóságos” programnyelvi elemekkel bővítették az SQL-t. Napjainkban több mint 50 SQL bázisú adatbáziskezelő nyelv létezik (a PC-ktől kezdve az egészen nagyszámítógépes rendszerekig). A nagygépes nyelvek között jelentősen felfutott a már említett DB2. A PC-kre az Ashton-Tate cég új dBase-t, a dBase IV-et fejlesztette ki (1988-ban), amely teljesen SQL-alapú. Elterjedt nyelvnek tekinthető a PC-kre írt PROFESSIONAL ORACLE nyelv, az XDB-SQL, az INGRES.

3. FEJEZET

AZ SQL ADATBÁZISKEZELŐ NYELV

Az IBM 1983-ban jelentette meg a DB2 nevű relációs adatbáziskezelő nyelvet, melynek alapját az 1976-ban kidolgozott

STRUCTURED QUERY LANGUAGE (SQL, stukturált lekérdező nyelv)

képezte. Az Amerikai Szabványügyi Hivatal (ANSI) 1986-ban a DB2 alapján szabványosította az SQL-t, létrehozva ezzel a STANDARD SQL-t.

Az SQL elsősorban lekérdező nyelv, mégpedig a relációkban tárolt információk visszanyerésének nyelve, de minimális adatbeviteli és módosítási lehetőségeket is nyújt.

Az SQL nem algoritmikus nyelv, tehát az algoritmikus nyelvekben megszokott vezérlőszervezetek (iteráció, szelekció) az SQL-ben explicite nem valósíthatók meg; az SQL az ilyen szervezeteket támogató nyelvekbe építhető be.

A továbbiakban a dBASE IV SQL lehetőségeit tárgyaljuk, amely a STANDARD SQL egy implementációja. Nem az SQL referenciaszerű bemutatása a célunk; elsősorban a nyelv logikai alapjainak megértését tűztük ki feladatul.

Az SQL parancsok szintaxisának ismertetése során az általánosan használt meta-nyelvet alkalmazzuk, amelynek elemei:

- | | |
|------------|---|
| • NAGYBETŰ | kötelező megadás, |
| • kisbetű | alkalmazó által történő megadás, |
| • [] | opcionális, |
| • ... | az előző szintaktikai elem ismétlődése, |
| • / | választás. |

Minden ezektől különböző karaktersorozat az SQL nyelv szintaxisához tartozik.

A dBASE IV SQL lehetőségeit először parancsmódban, interpreter alatt vizsgáljuk. Az SQL-interpreter úgy aktivizálható, hogy a dBASE IV interpreter aktív állapotában kiadjuk a

SET SQL ON

parancsot. Ezt követően az SQL-interpreter aktívvá válik, ezt az SQL. prompt jelzi.

3.1. Az SQL adatbázis fogalma és kezelésének parancsai

Az SQL különféle objektumokkal dolgozik. Ezen objektumokat fokozatosan fogjuk megismerni. Valamely alkalmazás objektumai alkotják az *adatbázist*. A dBASE IV SQL-ben az adatbázis önálló alkönyvtárként kerül kialakításra. Az objektumok egy részét az alkalmazó kezeli, másik részét a rendszer használja fel különböző adminisztrációs célokra. Ezeket *katalógustábláknak* (K-tábla) nevezzük.

3.1.1. Az adatbázis létrehozása

Az objektumok definiálása előtt az alkalmazónak létre kell hoznia az adatbázist.

Az adatbázist létrehozó parancs formátuma:

```
CREATE DATABASE adatbázisnév;
```

Példa: Hozzuk létre a vállalat nevű adatbázist.

Megoldás:

```
CREATE DATABASE vállalat;
```

Az adatbázis létrehozása után a katalógustáblák automatikusan definiálódnak és kezdeti értékekkel töltődnek fel. A létrehozott adatbázis megnyitottá válik.

3.1.2. Információ a létező adatbázisokról

Ha a meglévő adatbázisokról akarunk információkat megtudni, akkor a

```
SHOW DATABASE;
```

parancsot kell alkalmaznunk.

3.1.3. Adatbázis megnyitása

Meglévő adatbázist a

```
START DATABASE [adatbázisnév];
```

paranccsal nyithatunk meg. Ha nem adjuk meg az adatbázisnevet, akkor a megnyitott adatbázist lezárja.

3.1.4. Adatbázis lezárása

Az adatbázis lezárásához a

```
STOP DATABASE;
```

parancsot kell kiadnunk.

3.1.5. Adatbázis törlése

Adatbázis törlése a

```
DROP DATABASE adatbázisnév;
```

paranccsal érhető el.

A továbbiakban feltételezzük, hogy már létezik a vállalat nevű adatbázis és meg is van nyitva.

3.2. Adattábla definiálása

Az adatokat tartalmazó relációt *adattáblának* (A-tábla) nevezzük. Egy A-tábla akkor definiált, ha megadjuk:

- az oszlopok neveit és az
- oszlop értékének jellemzőit, az *oszlopparamétereket* vagy *adattípusokat*.

Az A-táblát definiáló parancs formátuma:

```
CREATE TABLE A-táblanév  
  (oszlopnév adattípus  
   [,oszlopnév adattípus]...);
```

dBASE IV SQL esetében az adattípus az alábbiak valamelyike lehet:

Adattípus	Értéktípus	Egészek száma	Tizedesek száma
SMALLINT	numerikus	6	0
INTEGER	numerikus	11	0
DECIMAL(x, y)	numerikus	$x + 1$	y
NUMERIC(x, y)	numerikus	x	y
FLOAT(x, y)	numerikus	x	y (a mantisszára)
CHAR(n)	karakteres	$n \leq 254$	
DATE	dátum	SET DATE ANSI opcióbeállítás javasolt, ekkor a megjelenítési kép: ÉÉ.HH.NN	
LOGICAL	logikai	.T.: igaz, .F.: hamis	

Példa: Definiáljuk a dolgozók nevű A-táblát. A tábla oszlopparaméterei legyenek a következők:

Oszlopnév	Adattípus	
azonosito	CHAR(3)	
nev	CHAR(20)	
fookod	CHAR(2)	– főosztálykód
okod	CHAR(2)	– osztálykód
alapfiz	SMALLINT	
bedatum	DATE	– belépési dátum
nemkod	LOGICAL	– .T.: hölgy, .F.: úr

Megoldás:

```
CREATE TABLE dolgozok  
  (azonosito CHAR(3),  
   nev       CHAR(20),  
   fookod    CHAR(2),  
   okod      CHAR(2),  
   alapfiz   SMALLINT,  
   bedatum   DATE,  
   nemkod    LOGICAL);
```


Feladat: Definiáljunk még három táblát, amelyeket a későbbiek során felhasználunk.

a) Definiáljuk a **fooszt** nevű A-táblát az alábbi oszlopparaméterekkel:

Oszlopnév	Adattípus	
fookod	CHAR(2)	
fomegn	CHAR(20)	- főosztály megnevezése

b) Definiáljuk az **oszt** nevű A-táblát az alábbi oszlopparaméterekkel:

Oszlopnév	Adattípus	
fookod	CHAR(2)	
okod	CHAR(2)	
omegn	CHAR(20)	- osztály megnevezése

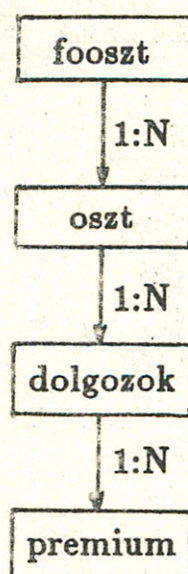
c) Definiáljuk a **premium** nevű A-táblát az alábbi oszlopparaméterekkel:

Oszlopnév	Adattípus	
azonosito	CHAR(3)	
osszeg	SMALLINT	- prémium összege

Ezzel négy relációt definiáltunk, ezekre írjuk most fel az adatmodellt.

Példa: Határozzuk meg az előbbieken definiált négy reláció közötti adatkapcsolatokat!

Megoldás: A kapcsolatokat az alábbi ábra szemlélteti:



A relációk között tehát többszintű hierarchikus kapcsolat van. 1:N azt jelenti, hogy minden **fooszt**-hoz több **oszt**, minden **oszt**-hoz több **dolgozok** s.í.t. tartozhat. A későbbiekben majd látjuk, hogy az SQL nem csak ilyen egyszerű adatmodellt képes kezelni.

Az A-tábláinkat definiáltuk, a K-táblák megfelelően módosított tartalma már ezt az állapotot tükrözi.

3.3. Új sorok elhelyezése az adattáblába

Az A-táblába helyezett új sor fizikailag a meglévő utolsó sor után tárolódik, de ennek a logikai kapcsolatok szempontjából nincs jelentősége.

Az új sor elhelyezésének egyik lehetséges parancsa:

```
INSERT INTO A-táblanév  
  [(oszlopnév-lista)]  
  VALUES (értéklista);
```

Ha az oszlopnév-lista (vesszővel elválasztott oszlopnevek) elmarad, akkor az A-tábla definiálásánál megadott oszlopnevek sorrendjében kell az értéklista elemeit megadni.

Példa: Töltsük fel a dolgozók A-táblát az alábbi adatokkal:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	8000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	13000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	10000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	7500	89.03.12	.T.

Megjegyzés: A dBASE IV-ben a karakteres konstanst ' vagy " jelek, a dátumkonstanst pedig {} közé kell tenni.

Megoldás: Az első sor tartalmának feltöltése az alábbi parancs hatására történik meg:

```
INSERT INTO dolgozok  
  VALUES ("002",  
    "Kis Pál",  
    "01",  
    "01",  
    12000,  
    {82.01.10},  
    .F.);
```

További sorokat az INSERT parancs ismételt kiadásával tudunk tárolni.

Ha meg akarjuk nézni a feltöltött A-tábla tartalmát, akkor ezt az alábbi paranccsal tehetjük meg:

```
SELECT *  
  FROM dolgozok;
```


A parancs végrehajtása után az alábbi táblázat jelenik meg:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	8000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	13000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	10000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	7500	89.03.12	.T.

(A SELECT parancsot később ismertetjük.)

Feladat: Töltsük fel a fooszt A-táblát az alábbi adatokkal:

FOOKOD	FOMEKN
01	01-es főosztály
02	02-es főosztály
03	03-as főosztály
04	04-es főosztály

Ellenőrizzük a feltöltött A-tábla tartalmát:

```
SELECT *
FROM fooszt;
```

A parancs végrehajtása után az alábbi táblázat jelenik meg:

FOOKOD	FOMEKN
01	01-es főosztály
02	02-es főosztály
03	03-as főosztály
04	04-es főosztály

Feladat: Töltsük fel az oszt A-táblát az alábbi adatokkal:

FOOKOD	OKOD	OMEKN
01	01	01 főoszt./01 oszt.
01	03	01 főoszt./03 oszt.
02	01	02 főoszt./01 oszt.
02	02	02 főoszt./02 oszt.
04	01	04 főoszt./01 oszt.
04	03	04 főoszt./03 oszt.

Ellenőrizzük a feltöltött A-tábla tartalmát:

```
SELECT *  
FROM oszt;
```

A parancs végrehajtása után az alábbi táblázat jelenik meg:

FOOKOD	OKOD	OMEGN
01	01	01 főoszt./01 oszt.
01	03	01 főoszt./03 oszt.
02	01	02 főoszt./01 oszt.
02	02	02 főoszt./02 oszt.
04	01	04 főoszt./01 oszt.
04	03	04 főoszt./03 oszt.

Feladat: Töltsük fel a premium A-táblát az alábbi adatokkal:

AZONOSITO OSSZEG

003	5000
020	2000
020	3000
003	7000
020	1000
007	2000
003	4000
021	5000

Ellenőrizzük a feltöltött A-tábla tartalmát:

```
SELECT *  
FROM premium;
```

A parancs végrehajtása után az alábbi táblázat jelenik meg:

AZONOSITO	OSSZEG
003	5000
020	2000
020	3000
003	7000
020	1000
007	2000
003	4000
021	5000

Az A-táblák feltöltése során csak konstansokat helyeztünk az *értéklista* elmeiként, az elem azonban dBASE IV-beli változónév vagy beépített függvény is lehet. A STANDARD SQL-ben nincs változó definiálására lehetőség!

3.4. Lekérdezés adattáblákból

Ezen funkció az SQL lelke, az SQL parancsok mindegyike ezt készíti elő. Látni fogjuk, hogy mindegyik lekérdezési parancs egy-egy felhasználói programot helyettesít.

A lekérdezés során *eredménytábla* (a továbbiakban: E-tábla) keletkezik, amely általában ideiglenes jellegű, azaz tartalma a megjelenítés hatására nem őrződik meg (ez felülbírálható).

A lekérdező parancs alparancsokból felépíthető összetett parancs, amelynek szerkezete a következő:

SELECT ...	<i>oszlop kiválasztása = projekció</i>
[INTO ...]	<i>az E-tábla 1. sorának tárolása</i>
FROM ...	<i>A-tábla nevek megadása = Descartes-szorzat</i>
[WHERE ...]	<i>sorok kiválasztása = szelekció</i>
[GROUP BY ...]	<i>csoportosítás</i>
[HAVING ...]	<i>csoportok közötti választás</i>
[UNION ...]	<i>E-táblák összefűzése = unió</i>
[ORDER BY ... / FOR UPDATE OF ...]	<i>E-tábla rendezése vagy A-tábla módosítása</i>
[SAVE TO TEMP ...];	<i>E-tábla megőrzése</i>

Az alparancsok megadásának sorrendje kötött, részletes ismertetésük előtt fontosnak tartjuk kiemelni, hogy végrehajtásuk az alábbi sorrendben történik:

FROM
WHERE
GROUP BY
HAVING
SELECT
UNION
ORDER BY
SAVE TO TEMP

Az E-tábla az alparancsok végrehajtása során, lépésenként alakul ki.

A dBASE IV SQL-interpreter alatti munka esetén javasoljuk a

SET PAUSE ON

opcióbeállítást, melynek következtében a betelt képernyőlap esetén egy billentyű leütésének hatására folytatódik az E-tábla megjelenítése.

A továbbiakban a lekérdező parancsban megadható alparancsokat ismertetjük.

3.4.1. A SELECT alparancs

Formátuma:

```
SELECT [ALL/DISTINCT] oszlopnév-lista /  
*
```

Az opciók hatását példákon keresztül mutatjuk be.

A * hatása: a FROM által kijelölt tábla minden oszlopa oszlopa az E-táblának is.

Példa: Jelenítsük meg a dolgozók A-tábla teljes tartalmát!

Megoldás:

```
SELECT *  
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	8000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	13000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	10000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	7500	89.03.12	.T.

Az oszlopnév-lista hatása: csak a lista elemei lesznek oszlopai az E-táblának.

A lista elemei származásuk szerint:

- a) a FROM által kijelölt tábla oszlopnevei, ha az oszlopnév duplikált, azaz több A-táblában is oszlopnév, akkor azt minősíteni kell az alábbiak szerint:

A-táblanév.oszlopnév

- b) a FROM által kijelölt tábla oszlopneveiből képzett kifejezés,
c) az aggregáló függvények valamelyike (lásd később),
d) egyéb kifejezés.

A b), c) és d) esetekben az E-tábla oszlopát *származtatott oszlopnak* nevezzük. Egyelőre csak azzal az esettel foglalkozunk, amikor az oszlopnév-lista eleme a FROM által kijelölt lista oszlopneveinek valamelyike.

Példa: Jelenítsük meg a dolgozók A-táblából a neveket és az alapfizetéseket.

Megoldás:

```
SELECT nev, alapfiz
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

NEV	ALAPFIZ
Kis Pál	12000
O Ibolya	8000
Réz Imre	15000
Nagy Ida	13000
Szabó Dénes	22000
Jó Péter	19000
Bár Noémi	10000
Kós János	20000
Piros Éva	7500

Az ALL hatása: az E-tábla duplikált sorokat is tartalmazhat. (Ez az alapértelmezés!)

Példa: Jelenítsük meg a dolgozók A-tábla összes fookod értékét.

Megoldás:

```
SELECT ALL fookod
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

FOOKOD
01
03
03
01
01
03
03
01
09

Megjegyzés: A megoldásból az ALL elmaradhat!

A DISTINCT hatása: az E-tábla a duplikált sorokat csak egyszer tartalmazza.

Példa: Jelenítsük meg a dolgozók A-tábla különböző fookod értékeit.

Megoldás:

```
SELECT DISTINCT fookod
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

FOOKOD
01
03

Példák származtatott oszlop megadására

a) Képezzük a dolgozók A-táblából az alábbi E-táblát:

Név 500 Ft-tal megnövelt alapfizetés

Megoldás:

```
SELECT nev, alapfiz+500
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

NEV	EXP1
Kis Pál	12500
O Ibolya	8500
Réz Imre	15500
Nagy Ida	13500
Szabó Dénes	22500
Jó Péter	19500
Bár Noémi	10500
Kós János	20500
Piros Éva	8000

b) Képezzük a dolgozók A-táblából az alábbi E-táblát:

Azonosító Kisbetűs név

Megoldás:

```
SELECT azonosito, LOWER(nev)
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

AZONOSITO	LOWER(NEV)
002	kis pál
005	o ibolya
003	réz imre
004	nagy ida
007	szabó dénes
009	jó péter
012	bár noémi
008	kós jános
010	piros Éva

c) Képezzük a dolgozók A-táblából az alábbi E-táblát:

Név Alapfizetés
 /hónap
 ...
 /hónap

1. megoldás:

```
SELECT nev, alapfiz, "/hónap"
FROM dolgozok;
```


A parancs hatására megjelenő E-tábla:

NEV	ALAPFIZ	EXP1
Kis Pál	12000	/hónap
O Ibolya	8000	/hónap
Réz Imre	15000	/hónap
Nagy Ida	13000	/hónap
Szabó Dénes	22000	/hónap
Jó Péter	19000	/hónap
Bár Noémi	10000	/hónap
Kós János	20000	/hónap
Piros Éva	7500	/hónap

2. megoldás:

```
megjegyzes=" /hónap"
SELECT nev,alapfiz,megjegyzes
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

NEV	ALAPFIZ	MEGJEGYZES
Kis Pál	12000	/hónap
O Ibolya	8000	/hónap
Réz Imre	15000	/hónap
Nagy Ida	13000	/hónap
Szabó Dénes	22000	/hónap
Jó Péter	19000	/hónap
Bár Noémi	10000	/hónap
Kós János	20000	/hónap
Piros Éva	7500	/hónap

Az a) és b) példa, valamint a c) első megoldásában az E-tábla fejlécsorát vizsgálva vegyük észre, hogy a származtatott oszlopok fejlécszövegei automatikusan képződnek. Egyes SQL-implementációkban lehetőség van a fejlécszöveg megadására is.

Példa: Képezzük a dolgozok A-táblából az alábbi E-táblát:

Nagybetűs név	Főosztálykód	Főosztálykód 15%-kal emelt alapfizetés
	Főosztálykód:	/hónap
	:	:
	Főosztálykód:	/hónap

Megoldás:

```
SELECT UPPER(nev),"Főosztálykód:",fokod,
        alapfiz*1.15,megjegyzes
FROM dolgozok;
```


A parancs hatására megjelenő E-tábla:

UPPER(NEV)	EXP1	FOOKOD	EXP2 MEGJEGYZES
KIS PÁL	Főosztálykód: 01		13800 /hónap
O IBOLYA	Főosztálykód: 03		9200 /hónap
RÉZ IMRE	Főosztálykód: 03		17250 /hónap
NAGY IDA	Főosztálykód: 01		14950 /hónap
SZABÓ DÉNES	Főosztálykód: 01		25300 /hónap
JÓ PÉTER	Főosztálykód: 03		21850 /hónap
BÁR NOÉMI	Főosztálykód: 03		11500 /hónap
KÓCS JÁNOS	Főosztálykód: 01		23000 /hónap
PIROS ÉVA	Főosztálykód: 03		8625 /hónap

Aggregáló függvények

A függvények értelmezési tartománya a közbülső E-tábla oszlopértékeinek halmaza, ennek oszlopértékeire alkalmazza az SQL az aggregáló függvényt, és a végső E-táblának a visszaadott egyetlen érték lesz része. Funkciójuknak megfelelően *gyűjtőfüggvényeknek* is nevezik ezen függvénytípusokat.

Röviden tekintsük át az SQL aggregáló függvényeit.

1. COUNT([DISTINCT] oszlopnév / *)

Megszámolja az oszlopnév által megadott oszlop elemeinek számát, és az értéket elhelyezi az E-táblában.

Példa: Mennyi a dolgozók A-tábla sorainak száma?

Megoldás:

```
SELECT COUNT(*)  
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

```
COUNT1  
9
```

Példa: Mennyi a dolgozók A-táblában a különböző főosztálykódok száma?

Megoldás:

```
SELECT COUNT(DISTINCT fookod)  
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

```
COUNT1  
2
```


2. SUM([ALL] kifejezés /
[ALL/DISTINCT] oszlopnev) : numerikus értékek összege

Összegezi a numerikus típusú oszlop értékeit, majd az összeget elhelyezi az E-táblába.

Példa: Mennyi a dolgozók A-táblában az alapfizetések összege?

Megoldás:

```
SELECT SUM(alapfiz)
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

SUM1
126500

3. A következő három aggregáló függvény paraméterezése megegyezik a SUM függvény paraméterezésével, a függvények mindegyike elhelyezi a visszaadott értéket az E-táblába.

- a) AVG(...) : átlag képzése
- b) MIN(...) : minimális érték keresése
- c) MAX(...) : maximális érték keresése

A MIN és MAX függvények argumentuma numerikus, dátum vagy karakteres típusú lehet.

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Alapfizetések átlaga	Legkisebb név	Legnagyobb belépési dátum
----------------------	---------------	---------------------------

Megoldás:

```
SELECT AVG(alapfiz),MIN(nev),MAX(bedatum)
FROM dolgozok;
```

A parancs hatására megjelenő E-tábla:

AVG1	MIN2	MAX3
14055.56	Bár Noémi	89.03.12

Az aggregáló függvények a SELECT vagy a HAVING alparancsokban adhatók meg. Ha a SELECT után állnak, akkor a SELECT-ben vagy csak aggregáló függvények szerepelhetnek, vagy csak olyan oszlopnevek, amelyek a GROUP BY alparancsban is adottak.

3.4.2. A FROM alparancs

Mint az előzőekben láttuk, az alparancs azon A-táblák megadására szolgál, amelyekből az E-táblát képezzük.

Formátuma:

FROM A-táblanév [másodnév][,A-táblanév [másodnév]]...

Egyelőre csak egyetlen A-táblából képzett E-táblákkal foglalkozunk, ekkor még a másodnév használata nem kellően indokolt.

3.4.3. A WHERE alparancs

Az alparancs megadásának hatására lesznek kiválasztva a FROM által meghatározott táblából azon sorok, amelyek majd az E-tábla sorainak kialakításában részt vesznek.

Formátuma:

WHERE feltétel

A feltétel egy logikai kifejezés, amely a FROM által meghatározott tábla minden sorára kiértékelődik, és igaz értéke esetén kerül a sor további feldolgozásra.

A feltétel elemei lehetnek:

- oszlopnevek
- beépített függvények
- változók
- konstansok
- műveleti jelek:
 - aritmetikai műveletek jelei: **, *, /, +, -
 - relációs műveleti jelek: <, <=, >, >=, =, <> vagy #
 - logikai műveletek jelei: NOT, AND, OR
- zárójelek
- aggregáló függvények (csak bizonyos esetekben)

Megjegyzés: A feltételben szereplő oszlopnévnek nem kötelező a SELECT alparancsban szerepelnie!

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Név Alapfizetés

azon női dolgozók soraiból, akiknek a fizetése > 9000 Ft.

Megoldás:

```
SELECT nev, alapfiz
FROM dolgozok
WHERE nemkod='T.' AND alapfiz>9000;
```


A parancs hatására megjelenő E-tábla:

NEV	ALAPFIZ
Nagy Ida	13000
Bár Noémi	10000

A feltétel egyszerűbb alakban való megadásához *predikátumfüggvények* használhatók.

A BETWEEN predikátum: zárt intervallum kijelölésére alkalmazható.
Formátuma:

kifejezés BETWEEN alsó érték AND felső érték

A predikátum igaz értéket vesz fel, ha:

alsó érték $\leq$ kifejezés $\leq$ felső érték

Az értékek és a kifejezés típusa dátum, karakteres vagy numerikus lehet.

Példa: Jelenítsük meg a *dolgozók* A-tábla értékeinek alapján az alábbi E-táblát:

Név	Belépési dátum
-----	----------------

azon dolgozókra, akik a 70-es években kezdtek dolgozni a vállalatnál.

Megoldás:

```
SELECT nev,bedatum
FROM dolgozok
WHERE bedatum BETWEEN {70.01.01} AND {79.12.31};
```

A parancs hatására megjelenő E-tábla:

NEV	BEDATUM
O Ibolya	78.07.08
Réz Imre	74.02.23
Bár Noémi	79.03.13

Az IN predikátum: értékegyezés vizsgálatára alkalmazható.
Formátuma:

kifejezés IN (értéklista)

A predikátum igaz értéket vesz fel, ha:

kifejezés = 1. listaelem OR kifejezés = 2. listaelem OR ...

Az értékek típusa dátum, karakteres vagy numerikus lehet.

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Azonosító	Név
azon dolgozókra, akiknek azonosítója 003, 007 vagy 010.	

Megoldás:

```
SELECT azonosito,nev
FROM dolgozok
WHERE azonosito IN ("003","007","010");
```

A parancs hatására megjelenő E-tábla:

AZONOSITO	NEV
003	Réz Imre
007	Szabó Dénes
010	Piros Éva

A LIKE predikátum: hasonlóság vizsgálatára alkalmazható.

Formátuma:

oszlopnév LIKE minta

A predikátum igaz értéket vesz fel, ha az oszlopnév konkrét értéke megfelel a mintának, azzal egyenlő. A mintában ún. *joker* (helyettesítő) karakterek is szerepelhetnek mégpedig:

- : az aláhúzás az adott pozíción egyetlen tetszőleges karaktert jelöl,
- % : a százalékjel tetszőleges számú karaktert helyettesít.

Az értékek típusa csak karakteres lehet.

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Név
azon dolgozókra, akiknek a nevében a második karakter i.

Megoldás:

```
SELECT nev
FROM dolgozok
WHERE nev LIKE "_i%";
```

A parancs hatására megjelenő E-tábla:

NEV
Kis Pál
Piros Éva

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Név

azon dolgozókra, akiknek a nevében i karakter szerepel.

Megoldás:

```
SELECT nev
FROM dolgozok
WHERE nev LIKE "%i%";
```

A parancs hatására megjelenő E-tábla:

NEV
Kis Pál
Bár Noémi
Piros Éva

A predikátumok önálló logikai kifejezések, logikai függvények, és mint ilyenek, részei lehetnek összetett logikai kifejezésnek.

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Név Alapfizetés

azon dolgozókra, akiknek az azonosítója 00-val kezdődik, és az alapfizetése a (9000, 18000) intervallumba esik.

Megoldás:

```
SELECT nev, alapfiz
FROM dolgozok
WHERE azonosito LIKE "00%"
AND
    alapfiz BETWEEN 9001 AND 17999;
```

A parancs hatására megjelenő E-tábla:

NEV	ALAPFIZ
Kis Pál	12000
Réz Imre	15000
Nagy Ida	13000

3.4.4. A GROUP BY alparancs

Az alparancs megadásával a közbülső E-tábla sorai azonos oszlopértékek alapján csoportokba szervezhetők. A végső E-táblában a csoportra vonatkozóan egy sor jelenik meg. A csoportra rendszerint aggregáló függvényt alkalmazunk.

Formátuma:

GROUP BY *oszlopnév* [, *oszlopnév*] ...

Az oszlopnévre viszonylag sok megkötés van. Ezek közül kimelten fontosak az alábbiak:

- minden SELECT-beli oszlopnévnek vagy a GROUP BY-ban, vagy a SELECT-ben felírt aggregáló függvény argumentumában kell szerepelnie,
- nem lehet származtatott oszlop neve (kivéve, ha SELECT-ben aggregáló függvény argumentumában szerepel).

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Főosztálykód	Osztálykód	Alapfizetések osztályösszege
--------------	------------	------------------------------

Megoldás:

```
SELECT fookod,okod,SUM(alapfiz)
FROM dolgozok
GROUP BY fookod,okod;
```

A parancs hatására megjelenő E-tábla tartalma:

G_FOOKOD	G_OKOD	SUM1
01	01	54000
01	02	13000
03	01	41500
03	02	18000

Megjegyzés: Az E-tábla nem lesz automatikusan rendezve a főosztálykódon belül osztálykódra!

3.4.5 A HAVING alparancs

A GROUP BY által kialakított közbülső E-táblából (= csoportokból) kiválasztja azon sorokat, amelyek részei lesznek a végleges E-táblának. Ha GROUP BY nincs megadva, akkor a FROM által meghatározott tábla egésze egy csoportnak tekintendő.

Formátuma:

HAVING *feltétel*

- Ha a feltétel összetett, akkor minden elemi feltételnek aggregáló függvényt kell tartalmaznia.
- A feltételben szereplő aggregáló függvényeknek a SELECT oszlopnév-listájában is szerepelniük kell.

Példa: Jelenítsük meg a dolgozók A-tábla értékei alapján az alábbi E-táblát:

Főosztálykód	Osztálykód	Osztály átlagfizetése
azon osztályokra, ahol az átlagfizetés > 12000 Ft.		

Megoldás:

```
SELECT fookod,okod,AVG(alapfiz)
FROM dolgozok
GROUP BY fookod,okod
HAVING AVG(alapfiz) > 12000;
```

A parancs hatására megjelenő E-tábla tartalma:

G_FOOKOD	G_OKOD	AVG1
01	01	18000
01	02	13000
03	01	13833.33

Az olyan SELECT parancsokat, amelyek felépítése

```
SELECT ...
FROM ...
[WHERE ...]
[GROUP BY ...]
[HAVING ...]
```

alakú, a továbbiakban al-SELECT-nek nevezzük.

3.4.6. Az UNION alparancs

Az alparancs hatására az E-tábla az al-SELECT-ek E-tábláinak konkatenálásával (egyesítésével) jön létre.

Formátuma:

```
al-SELECT UNION al-SELECT [UNION al-SELECT]...
```

Az al-SELECT-ek E-tábláinak kompatibiliseknek kell lenniük, azaz:

- azonos oszlopszámmal kell rendelkezniük,
- az oszlopok értékeinek azonos típusúaknak kell lenniük.

Az oszlopnevek eltérőek is lehetnek; azonos tartalmú sorok csak egyszer jelennek meg. Ha ORDER BY is szerepel, akkor abban csak sorszám adható meg rendezési szempontként.

Példa: Jelenítsük meg a dolgozok és a premium A-táblákban előforduló összes azonosító értékét.

Megoldás:

```
SELECT azonosito
FROM dolgozok
UNION
SELECT azonosito
FROM premium;
```


A parancs hatására megjelenő E-tábla tartalma:

AZONOSITO
002
003
004
005
007
008
009
010
012
020
021

3.4.7. Az ORDER BY alparancs

Az SQL az alparancs megadásának hatására rendezi az E-táblát.

Formátuma:

ORDER BY *oszlopnév / oszlop-sorszám* [ASC/DESC]
[,*oszlopnév / oszlop-sorszám* [ASC/DESC]]...

Megjegyzés:

- Az oszlopnévnek szerepelnie kell a SELECT-megadásban.
- Az oszlop-sorszám a SELECT-ben megadott oszlopnév-listára vonatkoztatott relatív sorszámmra utal.
- Az ASC növekvő (alapértelmezés), a DESC csökkenő rendezettségi irányt jelöl.
- A rendezési szempontok sorrendje egyben a rendezési hierarchiát is meghatározza.

Példa: Jelenítsük meg a dolgozók A-tábla értékei alapján az alábbi E-táblát:

Főosztálykód	Név	Alapfizetés
--------------	-----	-------------

mégpedig a főosztálykód növekvő sorrendjén belül az alapfizetés csökkenő sorrendjében.

Megoldás:

```
SELECT fookod,nev,alapfiz  
FROM dolgozok  
ORDER BY fookod,3 DESC;
```

Megjegyzés: ORDER BY után az alábbi megadások mindegyike helyes:

- 1,alapfiz DESC;
- fookod,alapfiz DESC;
- 1,3 DESC;

A parancs hatására megjelenő E-tábla tartalma:

FOOKOD	NEV	ALAPFIZ
01	Szabó Dénes	22000
01	Kós János	20000
01	Nagy Ida	13000
01	Kis Pál	12000
03	Jó Péter	19000
03	Réz Imre	15000
03	Bár Noémi	10000
03	O Ibolya	8000
03	Piros Éva	7500

3.4.8. A SAVE TO TEMP alparancs

Az alparancs megadása esetén az E-tábla ideiglenesen tárolódik (T-tábla), és az SQL-interpreter alatti munka idejéig A-táblaként kezelhető.

Formátuma:

```
SAVE TO TEMP T-táblanév  
  [(oszlopnév-lista)] [KEEP];
```

Megjegyzés:

- Az oszlopnév-lista megadással az E-tábla oszlopnevei átírhatók, ha a SELECT-ben származtatott oszlop van, akkor megadása kötelező.
- KEEP hatására T-táblanév.DBF kiterjesztésű állomány is keletkezik a DOS kurrens alkönyvtárban, ez nem SQL-tábla, hanem dBASE adatállomány, a T-táblával ellentétben nem szűnik meg.

Az INTO és FOR UPDATE OF alparancsokat később ismertetjük.

A lekérdező parancs helyett gyakran SELECT parancs elnevezést használnak.

3.5. Táblák összekapcsolása

A valós világ bonyolultabb annál, hogy adatainkat egyetlen táblába helyezhetnénk el, nem is beszélve a normalizálási megfontolások követelményeinek teljesítéséről. Az SQL szépsége a több táblában tárolt adatok visszanyerésére alkalmazott parancsok használatakor bontakozik ki igazán, ekkor válik érthetővé, hogy miért is tekinthetjük az SQL-t relációs adatbáziskezelő nyelvnek.

A táblák összekapcsolása azt jelenti, hogy a FROM mögött több A-tábla nevét adjuk meg. Ilyenkor a FROM kiértékelése során képezi a két vagy több reláció Descartes-szorzatát, és a keletkezett relációból a WHERE feltétel helyes megfogalmazásával választhatjuk ki a felhasználói igénynek megfelelő sorokat. (Ismételten felhívjuk a figyelmet arra, hogy a duplikált oszlopneveket az A-tábla nevével minősíteni kell.)

Példák a minősítésre:

- fooszt.fookod — a fooszt A-tábla fookod oszlopa
- dolgozok.* — a dolgozok A-tábla valamennyi oszlopa

Példa: Jelenítsük meg a dolgozok és fooszt A-táblák értékei alapján az alábbi E-táblát:

Név Főosztálykód (dolgozok-ból) Főosztálykód (fooszt-ból) Főosztály neve

Megoldás:

```
SELECT nev,dolgozok.fookod,fooszt.fookod,fomegn
FROM dolgozok,fooszt;
```

A parancs hatására megjelenő E-tábla tartalma:

DOLGOZOK→NEV	DOLGOZOK→FOOKOD	FOOSZT→FOOKOD	FOOSZT→FOMEKN
Kis Pál	01	01	01-es főosztály
Kis Pál	01	02	02-es főosztály
Kis Pál	01	03	03-as főosztály
Kis Pál	01	04	04-es főosztály
O Ibolya	03	01	01-es főosztály
O Ibolya	03	02	02-es főosztály
O Ibolya	03	03	03-as főosztály
O Ibolya	03	04	04-es főosztály
Réz Imre	03	01	01-es főosztály
Réz Imre	03	02	02-es főosztály
Réz Imre	03	03	03-as főosztály
Réz Imre	03	04	04-es főosztály
Nagy Ida	01	01	01-es főosztály
Nagy Ida	01	02	02-es főosztály
Nagy Ida	01	03	03-as főosztály
Nagy Ida	01	04	04-es főosztály
Szabó Dénes	01	01	01-es főosztály
Szabó Dénes	01	02	02-es főosztály
Szabó Dénes	01	03	03-as főosztály
Szabó Dénes	01	04	04-es főosztály
Jó Péter	03	01	01-es főosztály
Jó Péter	03	02	02-es főosztály
Jó Péter	03	03	03-as főosztály
Jó Péter	03	04	04-es főosztály
Bár Noémi	03	01	01-es főosztály
Bár Noémi	03	02	02-es főosztály
Bár Noémi	03	03	03-as főosztály
Bár Noémi	03	04	04-es főosztály
Kós János	01	01	01-es főosztály
Kós János	01	02	02-es főosztály
Kós János	01	03	03-as főosztály
Kós János	01	04	04-es főosztály
Piros Éva	03	01	01-es főosztály
Piros Éva	03	02	02-es főosztály
Piros Éva	03	03	03-as főosztály
Piros Éva	03	04	04-es főosztály

- A Descartes-szorzat képzése nyilvánvaló.
- A keletkezett E-tábla semmitmondó, értelmes információt nem tartalmaz.

A példa abnormalitásából is kiderülhetett, hogy a táblák összekapcsolása esetén kiemelt jelentősége van az értelmesen megfogalmazott WHERE-feltételnek. Szem előtt kell tartanunk viszont azt a tényt, miszerint a WHERE-feltétel már a Descartes-szorzat következtében keletkezett reláció soraiból szelektál! Ha a WHERE-feltétel olyan egyenlőség relációt tartalmaz, amelynek két oldalán különböző A-táblabeli, de azonos logikai tartalmú (esetleg azonos nevű is, ami természetes igény) oszlopnev szerepel, akkor a kapcsolat ún. *egyenlőségkapcsolat* (EQUIJOIN-kapcsolat).

Példa: Jelenítsük meg a dolgozók és a fooszt A-táblák értékei alapján az alábbi E-táblát:

Név Főosztálykód(dolgozók) Főosztálykód(fooszt) Főosztály megnevezése
és az E-tábla egy dolgozóra egy sort tartalmazzon, mégpedig annak főosztálykódjához adja meg a főosztály megnevezését.

Megoldás:

```
SELECT nev,dolgozok.fookod,fooszt.fookod,fomegn
FROM dolgozok,fooszt
WHERE dolgozok.fookod=fooszt.fookod;
```

A parancs hatására megjelenő E-tábla tartalma:

DOLGOZOK→NEV	DOLGOZOK→FOOKOD	FOOSZT→FOOKOD	FOOSZT→FOMEKN
Kis Pál	01	01	01-es főosztály
O Ibolya	03	03	03-as főosztály
Réz Imre	03	03	03-as főosztály
Nagy Ida	01	01	01-es főosztály
Szabó Dénes	01	01	01-es főosztály
Jó Péter	03	03	03-as főosztály
Bár Noémi	03	03	03-as főosztály
Kós János	01	01	01-es főosztály
Piros Éva	03	03	03-as főosztály

Megjegyzés: A főosztálykód oszlopa duplikált, redundáns.

Ha az EQUIJOIN jellegű táblakapcsolás esetén a relációban szereplő oszlopok közül az egyiket kihagyjuk az E-táblából, akkor a kapcsolat ún. *természetes kapcsolat* (NATURAL-JOIN kapcsolat). Az alkalmazások nagy részénél a táblák között ilyen jellegű kapcsolatokat kell megvalósítani.

Példa: Az előző feladat természetes kapcsolat típusú változata.

Megoldás:

```
SELECT nev,dolgozok.fookod,fomegn
FROM dolgozok,fooszt
WHERE dolgozok.fookod=fooszt.fookod;
```


A parancs hatására megjelenő E-tábla tartalma:

DOLGOZOK→NEV	DOLGOZOK→FOOKOD	FOOSZT→FOMEKN
Kis Pál	01	01-es főosztály
O Ibolya	03	03-as főosztály
Réz Imre	03	03-as főosztály
Nagy Ida	01	01-es főosztály
Szabó Dénes	01	01-es főosztály
Jó Péter	03	03-as főosztály
Bár Noémi	03	03-as főosztály
Kós János	01	01-es főosztály
Piros Éva	03	03-as főosztály

Példa: Jelenítsük meg a dolgozók, a fooszt és az oszt A-táblák értékei alapján az alábbi E-táblát:

Név Osztály megnevezése

és az E-tábla egy dolgozóra egy sort tartalmazzon.

Megoldás:

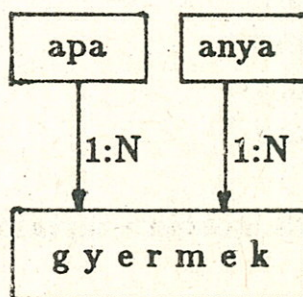
```
SELECT nev,omegn
FROM dolgozok,fooszt,oszt
WHERE dolgozok.fookod=fooszt.fookod AND
      dolgozok.okod=oszt.okod AND
      fooszt.fookod=oszt.fookod;
```

A parancs hatására megjelenő E-tábla tartalma:

DOLGOZOK→NEV	OSZT→OMEGN
Kis Pál	01 főoszt./01 oszt.
Szabó Dénes	01 főoszt./01 oszt.
Kós János	01 főoszt./01 oszt.

Megjegyzés: Vegyük észre, hogy nem minden dolgozóra vonatkozóan jelent meg eredmény sor.

A FROM hatására keletkezett Descartes-szorzatnál nincs jelentősége a relációk adatmodellben elfoglalt helyének, így nyilvánvaló, hogy az alábbi kapcsolatrendszert is össze tudnánk fogni egy SELECT-be:



Az A-táblát önmagával is összekapcsolhatjuk, ezt SELF-JOIN-nak nevezzük. Az összekapcsolás logikáját nem befolyásolja, hogy azonos vagy eltérő A-táblákra vonatkozik, mindenképpen a Descartes-szorzat alapján alakul ki a kapcsolt reláció. Amennyiben a táblát önmagával kapcsoljuk össze, duplikált oszlopnevek keletkeznek, így mindenképpen minősíteni kell azokat. A minősítéshez szükséges táblanév megadásához a FROM-ban kötelezően másodnevet kell definiálni.

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Főosztálykód Osztálykód Azonosító Név Kolléga neve

és az E-tábla legyen rendezett:

- főosztálykód
- osztálykód
- azonosító
- kolléga neve

oszlopokra, mindegyikre növekvő sorrendben.

Megoldás:

```
SELECT dol.fookod,dol.okod,dol.azonosito,dol.nev,dolg.nev
FROM dolgozok dol,dolgozok dolg
WHERE dol.fookod = dolg.fookod AND
      dol.okod = dolg.okod      AND
      dol.nev <# dolg.nev
ORDER BY dol.fookod,dol.okod,dol.azonosito,dolg.nev;
```

A parancs hatására megjelenő E-tábla tartalma:

DOL→FOOKOD	DOL→OKOD	DOL→AZONOSITO	DOL→NEV	DOLG→NEV
01	01	002	Kis Pál	Kós János
01	01	002	Kis Pál	Szabó Dénes
01	01	007	Szabó Dénes	Kis Pál
01	01	007	Szabó Dénes	Kós János
01	01	008	Kós János	Kis Pál
01	01	008	Kós János	Szabó Dénes
03	01	003	Réz Imre	Jó Péter
03	01	003	Réz Imre	Piros Éva
03	01	009	Jó Péter	Piros Éva
03	01	009	Jó Péter	Réz Imre
03	01	010	Piros Éva	Jó Péter
03	01	010	Piros Éva	Réz Imre
03	02	005	O Ibolya	Bár Noémi
03	02	012	Bár Noémi	O Ibolya

Megjegyzés: Az E-tábla csak azon dolgozókra tartalmaz sorokat, akiknek legalább egy kollégájuk van.

3.6. Beágyazott SELECT parancsok

A WHERE és a HAVING feltételében SELECT parancs is szerepelhet. Ekkor azt a SELECT-et, amely WHERE vagy HAVING feltételt tartalmaz, külső SELECT-nek, a WHERE, ill. HAVING feltételében szereplőt pedig belső vagy beágyazott SELECT-nek nevezzük. Az E-táblát a külső SELECT állítja össze a belső SELECT által szolgáltatott E-tábla értékeitől függően.

A dBASE IV SQL-ben csak olyan belső SELECT adható meg, amelynek E-táblája egy oszlop, és ez 0, 1 vagy több értékét tartalmaz.

A belső SELECT is tartalmazhat belső SELECT-et, vagyis a SELECT-ek egymásba ágyazhatók. A belső SELECT-et *egyszerű* al-lekérdezésnek nevezzük, ha annak WHERE feltételében nincs hivatkozás a külső SELECT FROM-jában megadott A-tábla oszlopára, ellenkező esetben a belső SELECT-et *kapcsolt* al-lekérdezésnek hívjuk. A belső SELECT nem tartalmazhat ORDER BY és UNION megadásokat; GROUP BY, ill. HAVING megadások lehetnek részei, viszont ezek a teljes SELECT-ben csak egyszer fordulhatnak elő. A belső SELECT-et mindig zárójelpárban kell megadni.

3.6.1 Egyszerű al-lekérdezés

Ilyen lekérdezés esetén először a belső SELECT értékelődik ki, majd ennek E-táblája alapján a külső SELECT állítja össze az E-táblát.

Két eset lehetséges: amikor a belső E-tábla egy, ill. amikor több értéket tartalmaz.

a) Egyetlen értéket tartalmazó belső E-tábla

Példa: Jelenítsük meg a dolgozók A-tábla értékeinek alapján az alábbi E-táblát:

Név Alapfizetés
azon dolgozókra, akiknek alapfizetése < átlagfizetés.
A tábla legyen a nevek szerint rendezve.

Megoldás:

```
SELECT nev, alapfiz
FROM dolgozok
WHERE alapfiz <
      (SELECT AVG(alapfiz)
       FROM dolgozok)
ORDER BY nev;
```

A parancs hatására megjelenő E-tábla tartalma:

NEV	ALAPFIZ
Bár Noémi	10000
Kis Pál	12000
Nagy Ida	13000
O Ibolya	8000
Piros Éva	7500

Példa: Jelenítsük meg a dolgozók A-tábla értékei alapján az alábbi E-táblát:

Név Alapfizetés
azon dolgozókra, akiknek alapfizetése az alábbi intervallumba esik:
(átlagfizetés-4000,átlagfizetés+4000)
A tábla legyen névre rendezett.

Megoldás:

```
SELECT nev,alapfiz
FROM dolgozok
WHERE alapfiz >
      (SELECT AVG(alapfiz)-4000
       FROM dolgozok)
      AND
      alapfiz <
      (SELECT AVG(alapfiz)+4000
       FROM dolgozok)
ORDER BY nev;
```

A parancs hatására megjelenő E-tábla tartalma:

NEV	ALAPFIZ
Kis Pál	12000
Nagy Ida	13000
Réz Imre	15000

b) Több értéket tartalmazó belső E-tábla

Ha belső SELECT több értéket szolgáltat, akkor a feltételben kötelezően használni kell az IN, ANY vagy ALL predikátumok valamelyikét.

A predikátumok megadásának formátuma:

predikátum (belső SELECT)

1. Az IN predikátum: értékegyezést vizsgál.

Igaz értéket ad vissza, ha az = reláció valamelyik belső E-táblabeli értékre igaz.

Példa: Jelenítsük meg a dolgozók és főoszt A-táblák értékeinek alapján az alábbi E-táblát:

Főosztálykód Főosztály megnevezése
azon főosztályokra, amelyeknek nincs alkalmazottja.

Megoldás:

```
SELECT fookod,fomegn
FROM fooszt
WHERE fookod NOT IN (SELECT fookod
                     FROM dolgozok);
```

A parancs hatására megjelenő E-tábla tartalma:

FOOKOD	FOMEKN
02	02-es főosztály
04	04-es főosztály

2. Az *ANY* predikátum: a reláció teljesülését vizsgálja.

Igaz értéket ad vissza, ha a megadott reláció valamelyik belső E-táblabeli értékre igaz.

Példa: Mi lesz az alábbi SELECT parancs E-táblája?

```
SELECT fookod,nev
FROM dolgozok
WHERE fookod < ANY (SELECT fookod
                   FROM fooszt
                   WHERE fookod NOT
                     IN ("03","04"));
```

Megoldás:

FOOKOD	NEV
01	Kis Pál
01	Nagy Ida
01	Szabó Dénes
01	Kós János

3. Az *ALL* predikátum: a reláció teljesülését vizsgálja.

Igaz értéket ad vissza, ha a megadott reláció valamennyi belső E-táblabeli értékre igaz.

Példa: Mi lesz az alábbi SELECT parancs E-táblája?

```
SELECT fookod,nev
FROM dolgozok
WHERE fookod <= ALL (SELECT fookod
                   FROM fooszt);
```

Megoldás:

FOOKOD	NEV
01	Kis Pál
01	Nagy Ida
01	Szabó Dénes
01	Kós János

3.6.2. Kapcsolt al-lekérdezés

Ekkor a lekérdezés végrehajtása során a külső SELECT egyenként adja át a FROM által meghatározott tábla sorait a belső SELECT-nek, amely az átadott sorra kiértékelődik, és ennek E-táblája alapján a külső SELECT állítja össze a végső E-táblát. Ha a külső és a belső SELECT azonos A-táblát használ, akkor mindkettő FROM-jában másodnevet kell definiálni.

Ekkor is két eset lehetséges.

a) Egy értéket tartalmazó belső E-tábla

Végrehajtásának logikája:

1. A külső SELECT átad egy sort a belső SELECT-nek.
2. Erre a sorra a belső SELECT kiértékelődik, visszaad egy értéket.
3. Ha ezen értékre a külső SELECT WHERE vagy HAVING feltétele teljesül, akkor a végső E-tábla kialakításához veszi a külső SELECT aktuális sorát.
4. A külső SELECT átadja a következő sort, és a végrehajtás a 2. pontban folytatódik.

Példa: Jelenítsük meg a dolgozók A-tábla értékei alapján az alábbi E-táblát:

Főosztálykód Osztálykód Név Alapfizetés

azon dolgozókra, akiknek alapfizetése a szervezeti egységen (főosztály, osztály) belül a legnagyobb.

Az E-tábla legyen főosztálykódra és osztálykódra növekvőleg rendezett.

Megoldás:

```
SELECT fookod,okod,nev,alapfiz
FROM dolgozok dol
WHERE alapfiz = (SELECT MAX(alapfiz)
                  FROM dolgozok dolg
                  WHERE dol.fookod=dolg.fookod AND
                    dol.okod=dolg.okod)
ORDER BY fookod,okod;
```

A parancs hatására megjelenő E-tábla tartalma:

DOL→FOOKOD	DOL→OKOD	DOL→NEV	DOL→ALAPFIZ
01	01	Szabó Dénes	22000
01	02	Nagy Ida	13000
03	01	Jó Péter	19000
03	02	Bár Noémi	10000

Alkalmazható az EXISTS predikátum, amely akkor ad vissza igaz értéket, ha a belső SELECT E-táblája nem üres. EXISTS predikátum használata esetén, és csak ekkor, a belső SELECT-ben az oszlopnév-lista helyén * is megadható.

Példa: Jelenítsük meg a dolgozók és a premium A-táblák értékeinek alapján az alábbi E-táblát:

Azonosító Név
azon dolgozókra, akik legalább egyszer prémiumot kaptak.

Megoldás:

```
SELECT azonosito,nev
FROM dolgozok
WHERE EXISTS
      (SELECT *
       FROM premium
       WHERE dolgozok.azonosito=premium.azonosito);
```

A parancs hatására megjelenő E-tábla tartalma:

DOLGOZOK→AZONOSITO	DOLGOZOK→NEV
003	Réz Imre
007	Szabó Dénes

Megjegyzés: Az előző E-tábla EXISTS nélkül is lehozható:

```
SELECT azonosito,nev
FROM dolgozok
WHERE azonosito IN
      (SELECT azonosito
       FROM premium);
```

A parancs hatására megjelenő E-tábla tartalma:

AZONOSITO	NEV
003	Réz Imre
007	Szabó Dénes

Ez többértékű egyszerű belső SELECT-re vezeti vissza a megoldást.

b) Több értéket tartalmazó belső E-tábla

Végrehajtásának logikája:

1. A külső SELECT átad egy sort a belső SELECT-nek.
2. Erre a sorra a belső SELECT kiértékelődik, visszaadja az E-táblájának első értékét.
3. Ha ezen értékre a külső SELECT WHERE vagy HAVING feltétele teljesül, akkor a végső E-tábla kialakításánál a külső SELECT aktuális sorát figyelembe veszi.
4. A belső SELECT visszaadja E-táblájának következő értékét, és folytatódik a 3. pontban.
5. A külső SELECT átadja a következő sort és folytatódik a 2. pontban.

Példa: Jelenítsük meg a dolgozók és premium A-táblák értékei alapján az alábbi E-táblát:

Azonosító Név

azon dolgozókra, akik legalább egyszer prémiumot kaptak, és egy dolgozóra annyi sort tartalmazzon az E-tábla, ahányszor prémiumot kapott. Az E-tábla legyen az azonosító szerint növekvő sorrendben rendezett.

Megoldás:

```
SELECT azonosito,nev
FROM dolgozok
WHERE azonosito =
      (SELECT azonosito
       FROM premium
       WHERE dolgozok.azonosito=premium.azonosito)
ORDER BY azonosito;
```

A parancs hatására megjelenő E-tábla tartalma:

DOLGOZOK→AZONOSITO	DOLGOZOK→NEV
003	Réz Imre
003	Réz Imre
003	Réz Imre
007	Szabó Dénes

A belső SELECT-tel foglalkozó rész zárásaként felhívjuk a figyelemet arra, hogy a feladatok gyakran belső SELECT-tel és táblakapcsolással is megoldhatók. Ekkor az utóbbi megoldás a javasolt, ez áttekinthetőbb parancsot eredményez.

3.7. Al-SELECT használata más parancsokban

Emlékeztetőül: al-SELECT-nek neveztük azt a SELECT-t, amelynek felépítése:

```
SELECT ...
FROM ...
[WHERE ...]
[GROUP BY ...]
[HAVING ...]
```

A fejezet további részében feltételezzük, hogy az A-táblák az eredeti tartalommal rendelkeznek, és a parancsok ezeket fogják folytonosan módosítani, azaz a módosítás az esetlegesen egy előző módosítás következtében létrejött táblatartalmat érinti, nem térünk vissza az eredeti táblatartalmakhoz.

A módosítások a dolgozók A-táblát érintik, ennek kezdeti tartalma:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	8000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	13000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	10000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	7500	89.03.12	.T.

Al-SELECT az alábbi esetekben használható:

1. WHERE, ill. HAVING feltételében belső SELECT-ként. (Ezzel foglalkoztunk a tankönyv előző részében.)
2. Új sorok beszúrása

Eddig új sor tárolására az

```
INSERT INTO A-táblanév
  [(oszlopnév-lista)]
  VALUES (értéklista);
```

parancsot használtuk.

Al-SELECT használatával lehetőség van több új sor elhelyezésére.

A parancs formátuma:

```
INSERT INTO A-táblanév
  [(oszlopnév-lista)]
  al-SELECT;
```

Ebben az esetben az új sorok értékeit az al-SELECT E-táblájának soraiból tölti fel az SQL.

Példa: A dolgozók A-táblába vigyük be azon premium A-táblabeli azonosítókat, amelyek nagyobbak 019-nél.

Megoldás:

```
INSERT INTO dolgozok
  (azonosito)
  SELECT DISTINCT azonosito
  FROM premium
  WHERE azonosito > "019";
```


A parancs eredményeként a dolgozók A-tábla tartalma az alábbi lesz:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	8000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	13000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	10000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	7500	89.03.12	.T.
020					.	.F.
021					.	.F.

Megjegyzés:

- Megfontolandó a DISTINCT szükségessége!
- Az új sorok nem feltöltött oszlopai típus-üres kezdeti értéket kapnak.

3. Meglévő sorok tartalmának módosítása

A parancs formátuma:

```
UPDATE A-táblanév
SET oszlopnév = kifejezés
[,oszlopnév = kifejezés]...
[WHERE feltétel];
```

Példa: A dolgozók A-táblában szereplő nők alapfizetését növeljük meg 1000 Ft-tal.

Megoldás:

```
UPDATE dolgozok
SET alapfiz = alapfiz + 1000
WHERE nemkod=.T.;
```

A parancs eredményeként a dolgozók A-tábla tartalma az alábbi lesz:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	9000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	14000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	11000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	8500	89.03.12	.T.
020					.	.F.
021					.	.F.

Al-SELECT használata esetén a WHERE feltételében megadott belső SELECT E-tábla értékei jelölik ki a módosítandó sorokat.

Példa: A dolgozók A-táblába az előbbieken bevitt új sorok alapfizetési oszlopát módosítsuk 30000-re.

Megoldás:

```
UPDATE dolgozok
SET alapfiz = 30000
WHERE azonosito IN (SELECT DISTINCT azonosito
FROM premium
WHERE azonosito > "019");
```

A parancs eredményeként a dolgozók A-tábla tartalma az alábbi lesz:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
002	Kis Pál	01	01	12000	82.01.10	.F.
005	O Ibolya	03	02	9000	78.07.08	.T.
003	Réz Imre	03	01	15000	74.02.23	.F.
004	Nagy Ida	01	02	14000	69.11.24	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	11000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	8500	89.03.12	.T.
020				30000	.	.F.
021				30000	.	.F.

4. Meglévő sorok fizikai törlése

A parancs formátuma:

```
DELETE
FROM A-táblanév
[WHERE feltétel];
```

Példa: A dolgozók A-táblából töröljük azon dolgozók adatait, akiknek az azonosítója kisebb 005-nél.

Megoldás:

```
DELETE
FROM dolgozok
WHERE azonosito < "005";
```

A parancs eredményeként a dolgozók A-tábla tartalma az alábbi lesz:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
005	O Ibolya	03	02	9000	78.07.08	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	11000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	8500	89.03.12	.T.
020				30000	.	.F.
021				30000	.	.F.

Al-SELECT használata esetén a WHERE feltételében megadott belső SELECT E-tábla értékei jelölik ki a törlendő sorokat.

Példa: A dolgozók A-táblából töröljük a 2. pontban bevitt új sorokat.

Megoldás:

DELETE

FROM dolgozók

WHERE azonosito IN (SELECT DISTINCT azonosito
FROM premium
WHERE azonosito > "019");

A parancs eredményeként a dolgozók A-tábla tartalma az alábbi lesz:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
005	O Ibolya	03	02	9000	78.07.08	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	11000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	8500	89.03.12	.T.

5. Virtuális tábla (V-tábla) létrehozása

Virtuális vagy nézettáblának (V-tábla) azt a táblát nevezzük, amely mögött ugyan ténylegesen, fizikailag nem áll adattartalom, mégis A-táblaként kezelhető.

A V-táblát létrehozó parancs formátuma:

CREATE VIEW *V-táblanév*

[*oszlopnév-lista*]

AS *al-SELECT*

[WITH CHECK OPTION];

Az utasítás hatására az SQL tárolja a V-tábla létrehozásának paramétereit, és a V-táblára való hivatkozás során hajtja végre a paraméterek által definiált V-tábla generálását.

Az *al-SELECT FROM*-jában egy vagy több A- vagy V-tábla neve adható meg. Általában az mondható el, hogy minden olyan esetben, amikor A-táblára történik hivatkozás, ott V-tábla neve is megadható.

A V-tábla ún. *UPDATE V-tábla*, ha a V-táblán keresztül végzett módosítás (vagyis ha a táblatartalmat módosító parancsban V-táblanevet adunk meg) érinti azt az A-táblát is, amelyre alapozva a V-táblát létrehoztuk. Ilyen típusú V-tábla létrehozásának minimális feltétele, hogy vagy A-táblára, vagy *UPDATE V-táblára* legyen alapozva.

A *WITH CHECK OPTION* megadása esetén meglévő sor módosítása vagy új sor bevitele csak akkor lehetséges az A-táblába a V-táblán keresztül, ha az értékek eleget tesznek a V-táblát létrehozó *al-SELECT WHERE*-feltételének.

V-tábla létrehozásának indokai:

- gyakran szükséges E-tábla „tartalmának” tárolása
- A-tábla oszlopainak, sorainak csökkentése, elrejtése bizonyos felhasználók előtt
- A-tábla stuktúrájának átszervezése

Példa: A dolgozók A-táblára alapozva hozzuk létre az alábbi, nok nevű V-táblát:

nok_nevei fizetes

A létrehozás után jelenítsük meg a tábla „tartalmát”.

Megoldás:

```
CREATE VIEW nok
(nok_nevei,fizetes)
AS SELECT nev,alapfiz
FROM dolgozok
WHERE nemkod=.T.;
```

```
SELECT *
FROM nok;
```

A parancs hatására megjelenő E-tábla tartalma:

NOK_NEVEI	FIZETES
O Ibolya	9000
Bár Noémi	11000
Piros Éva	8500

Példa: A dolgozók A-táblára alapozva hozzuk létre az alábbi, alk nevű V-táblát:

azonosito nev szervezet

ahol a szervezet a főosztálykód és az osztálykód konkatenációja. (A konkatenáció műveletének jele: +, hatására összefűzi a két operandust.)

A létrehozás után jelenítsük meg a tábla „tartalmát”.

Megoldás:

```
CREATE VIEW alk
(azonosito,nev,szervezet)
AS SELECT azonosito,nev,fookod+okod
FROM dolgozok;
```

```
SELECT *
FROM alk;
```

A parancs hatására megjelenő E-tábla tartalma:

AZONOSITO	NEV	SZERVEZET
005	O Ibolya	0302
007	Szabó Dénes	0101
009	Jó Péter	0301
012	Bár Noémi	0302
008	Kós János	0101
010	Piros Éva	0301

Példa: A nok V-táblára alapozva hozzuk létre az alábbi, kisfizm nevű V-táblát:

nevek kisber

a nok V-tábla azon soraiból, ahol a fizetes < 10000 Ft. A létrehozás után jelenítsük meg a tábla „tartalmát”.

Megoldás:

```
CREATE VIEW kisfizm
(nevek,kisber)
AS SELECT *
FROM nok
WHERE fizetes < 10000;
```

```
SELECT *
FROM kisfizm;
```

A parancs hatására megjelenő E-tábla tartalma:

NEVEK	KISBER
O Ibolya	9000
Piros Éva	8500

Példa: A dolgozok és fooszt A-táblákra alapozva hozzuk létre az alábbi aznefokm nevű V-táblát:

azon nev fkod fmegn

amely V-tábla egy sora egy dolgozóra vonatkozóan tartalmazza az azonosítót, nevet, főosztálykódot és a főosztály megnevezését.

A létrehozás után jelenítsük meg a tábla „tartalmát”.

Megoldás:

```
CREATE VIEW aznefokm
(azon,nev,fkod,fmegn)
AS SELECT azonosito,nev,dolgozok.fookod,fomegn
FROM dolgozok,fooszt
WHERE dolgozok.fookod=fooszt.fookod;
```

```
SELECT *
FROM aznefokm;
```

A parancs hatására megjelenő E-tábla tartalma:

AZON	NEV	FKOD	FMEGN
005	O Ibolya	03	03-as főosztály
007	Szabó Dénes	01	01-es főosztály
009	Jó Péter	03	03-as főosztály
012	Bár Noémi	03	03-as főosztály
008	Kós János	01	01-es főosztály
010	Piros Éva	03	03-as főosztály

3.8. Másodlagos táblanevek

Esetenként javasolt az A- vagy V-táblának másodlagos nevet adni, például akkor, ha már valamilyen néven létezik, és saját alkalmazásunkban rövidebb vagy beszédesebb néven szeretnénk hivatkozni rá. A másodlagos név nem tévesztendő össze a SELECT parancs FROM-jában megadható másodnévvel!

A másodlagos nevet definiáló parancs formátuma:

```
CREATE SYNONYM másodlagos név  
FOR táblanév ;
```

A másodlagos név minden olyan helyen megadható, ahol táblanévre történik hivatkozás.

3.9. Indextáblák

Az SQL az optimalizált feldolgozást saját algoritmusát alkalmazva valósítja meg. Ez általában nem ismert, viszont a leírások tartalmaznak javaslatokat arra vonatkozóan, hogy az algoritmus miképpen gyorsítható. Például ha egy A-tábla tartalmát gyakran szeretnénk valamilyen rendezettségben látni, akkor célszerű a megfelelő oszlopokra indextáblát (I-tábla) definiálni. Az indexelés után az I-tábla rendezetten tartalmazza az érintett oszlopok értékeit.

Az I-táblát definiáló parancs formátuma:

```
CREATE [UNIQUE] INDEX I-táblanév  
ON A-táblanév  
(oszlopnév [ASC/DESC])  
[,oszlopnév [ASC/DESC]]...;
```

Megjegyzés:

- ASC és DESC egy parancson belül nem keverhető.
- UNIQUE megadásának hatására az A-tábla tartalmának módosítása esetén megtiltja az I-tábla definiálását meghatározó oszlopok értékeinek duplikálását.

Az SQL-nek nincs olyan parancsa, amely az I-tábla használatát explicite kezdeményezné!

3.10. Táblák törlése

A táblák törlését az alábbi parancsok hajtják végre:

- a) DROP INDEX *I-táblanév* ;
- b) DROP SYNONYM *másodlagos név* ;
- c) DROP TABLE *A-táblanév* ;
- d) DROP VIEW *V-táblanév* ;

Megjegyzés:

- A törölt táblára történő hivatkozás a hibát okoz.
- A másodlagos név törlése csak azt jelenti, hogy ilyen névvel a tábla nem azonosítható. A tábla az eredeti vagy a még létező másodlagos néven keresztül továbbra is kezelhető.

3.11. SQL parancsok beágyazása programnyelvekbe

Az SQL parancs beágyazása azt jelenti, hogy a parancs valamely befogadó nyelven írt felhasználó programban van elhelyezve. Ilyen esetben általában előfordító (preprocesszor) alakítja át a befogadó nyelvnek megfelelő eljárashívás formájára.

A dBASE IV speciálisan oldja meg a beágyazást: az SQL parancsot tartalmazó forrásprogram kötelező kiterjesztése .PRS, és az ilyen program aktiválása után automatikusan átkapcsol SQL módba. Az SQL parancsot programban elhelyezve is ; -vel kell lezárni.

Az SQL parancs lefutásának helyessége predefiniált változók tartalmának lekérdezése alapján vizsgálható. Az egyik ilyen változó az SQLCODE, melynek 0 értéke jelzi a hiba nélküli végrehajtást. Másik változó az SQLCNT, amely az SQL parancs által érintett tábla sorainak számát tartalmazza. Mindkét változó dBASE IV parancsmódban is lekérdezhető.

A beágyazásra vonatkozó SQL eszközöket és logikát példákon keresztül mutatjuk be, ezek mindegyikét a dBASE IV programnyelvén írtuk. (Részletesebb magyarázatot csak az SQL-t érintő utasításokhoz adunk.)

A példákban feltételezzük, hogy az SQL-interpreter az aktív (ezt az SQL. prompt mutatja), és hogy a vállalat adatbázis nyitott.

A dolgozók A-tábla induló tartalma legyen:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
005	O Ibolya	03	02	9000	78.07.08	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	11000	79.03.13	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	8500	89.03.12	.T.

SQL-interpreter alatt is kiadható SQL parancs hívása

Feladat: azon dolgozók A-táblabeli sorok megjelenítése, amelyekben a főosztálykód a billentyűzetről beolvasott értékkel egyenlő.

Megoldás:

SQL. MODIFY COMMAND flista.prs && A dBASE IV-szövegszerkesztő aktivizálása.

```
*****
*
* flista.prs: 1. példa
*
*          SQL-interpreter alatt is kiadható SQL parancs hívása
*
*****
```

CLEAR

ACCEPT "Kérem a főosztálykód értékét:" TO wfokod

?

?

SELECT *

FROM dolgozok

WHERE fokod=wfokod;

?

?

WAIT

CLEAR

*

* Program vége.

SQL. DO flista

&& A program futtatása.

A program futásának képernyőképe:

Kérem a főosztálykód értékét: 03

AZONOSITO	NEV	FOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
005	O Ibolya	03	02	9000	78.07.08	.T.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	11000	79.03.13	.T.
010	Piros Éva	03	01	8500	89.03.12	.T.

Press any key to continue...

E-tábla első vagy egyetlen sorértékeinek mentése változókbá

(Csak a beágyazott SELECT-ben megadható INTO alparancs hatására valósítható meg.)

Feladat: a dolgozók A-tábla 1. sorából a név és az alapfizetés megjelenítése.

Megoldás:

SQL. MODIFY COMMAND egysor.prs && A dBASE IV-szövegszerkesztő aktiválása

```
*****
*
* egysor.prs: 2. példa
*
*           E-tábla első vagy egyetlen sor-értékeinek mentése változókbá
*
*****
```

CLEAR

?

?

- * Az INTO alparancs hatására a wnev, ill. a walapfiz nevű változók
- * felveszik az E-tábla első sorának nev és alapfiz oszlopértékeit.

SELECT nev,alapfiz

INTO wnev,walapfiz

FROM dolgozok;

IF SQLCNT > 0

&& Ha nem üres az E-tábla.

? "Név:",wnev

? "Alapfizetés:",walapfiz

ENDIF

?

WAIT

CLEAR

*

* Program vége.

SQL. DO egysor

&& A program futtatása.

A program futásának képernyőképe:

Név: 0 Ibolya

Alapfizetés: 9000

Press any key to continue...

E-tábla soronkénti feldolgozása

(Így csak programban oldható meg.)

Feladat: azon dolgozók A-táblabeli név és alapfizetés értékeinek soronkénti megjelenítése, amelyekben a főosztálykód 03.

Megoldás:

SQL. MODIFY COMMAND lista03.prs && A dBASE IV-szövegszerkesztő aktiválása.

```
* * * * *
* lista03.prs: 3. példa
*
*           E-tábla soronkénti feldolgozása
*
* * * * *
```

CLEAR

* Fejlécsorok megjelenítése.

?

? "Név",SPACE(20),"Alapfizetés"

? "----",SPACE(20),"-----"

?

* Kurzor definiálása; ez mutat majd az E-tábla soraira (egyenként).

DECLARE dlist CURSOR

FOR SELECT nev,alapfiz

FROM dolgozók

WHERE fookod="03";

* Kurzor nyitása, ekkor hajtódik végre a SELECT, és a kurzor az E-tábla

* első sora elé mutat.

OPEN dlist;

* Feldolgozási ciklus.

DO WHILE .T. && Elvileg végtelen ciklus.

* A kurzor átállítása az E-tábla következő sorára, a sor értékeinek

* átvitele változókba.

FETCH dlist INTO wnev,walapfiz;

IF SQLCODE # 0

EXIT

&& Ha az E-tábla végére értünk, akkor vége a

ENDIF

&& ciklusnak.

? wnev,SPACE(4),STR(walapfiz,6) && E-tábla sorának feldolgozása.

ENDDO

* Kurzor lezárása.

CLOSE dlist;

?

?

WAIT

CLEAR

*

* Program vége.

SQL. DO lista03

&& Program futtatása.

A program futásának képernyőképe:

Név	Alapfizetés
----	-----
O Ibolya	9000
Jó Péter	19000
Bár Noémi	11000
Piros Éva	8500

Press any key to continue...

A-tábla kurzor által kijelölt sorainak módosítása

(Így csak programban oldható meg.)

Feladat: azon dolgozók A-táblabeli sorok alapfizetésének 30000-re, ill. belépési dátumának a rendszerdátumra való módosítása, ahol az azonosító > 009.

Megoldás:

SQL. MODIFY COMMAND kmodosit.prs && A dBASE IV-szövegszerkesztő aktiv.

```
*****
*
* kmodosit.prs: 4. példa
*
*           A-tábla kurzor által kijelölt sorainak módosítása
*
*****
```

* Kurzor definiálása; ez mutat majd az E-tábla soraira (egyenként).

* A kurzort most módosításra kell definiálni.

DECLARE dk CURSOR

FOR SELECT azonosito

FROM dolgozok

WHERE azonosito > "009"

FOR UPDATE OF alapfiz,bedatum;

* Kurzor nyitása, ekkor hajtódik végre a SELECT és a kurzor az E-tábla

* első sora elé mutat.

OPEN dk;

* Feldolgozási ciklus.

DO WHILE .T. && Elvileg végtelen ciklus.

* A kurzor átállítása az E-tábla következő sorára, a sor értékeinek

* átvitele változókba.

FETCH dk INTO wazonosito;

IF SQLCODE # 0

EXIT

&& Ha az E-tábla végére értünk. akkor vége a

ENDIF

&& ciklusnak.

- * Az A-tábla sorának módosítása. Azt a sort, amelyet módosítani kell,
- * az E-táblának azon sorértéke határozza meg, amelyre a kurzor mutat.

UPDATE dolgozok

SET alapfiz=30000,

bedatum=DATE()

WHERE CURRENT OF dk;

ENDDO

- * Kurzor lezárása.

CLOSE dk;

*

- * Program vége.

SQL. DO kmodosit

&& Program futtatása.

A program lefutásának eredményeként a dolgozok A-tábla tartalma:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
005	O Ibolya	03	02	9000	78.07.08	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
012	Bár Noémi	03	02	30000	91.01.05	.T.
008	Kós János	01	01	20000	81.07.12	.F.
010	Piros Éva	03	01	30000	91.01.05	.T.

A-tábla kurzor által kijelölt sorainak törlése

(Így csak programban oldható meg.)

Feladat: azon dolgozok A-táblabeli sorok törlése, ahol az azonosító > 009.

Megoldás:

SQL. MODIFY COMMAND ktorol.prs && A dBASE IV-szövegszerkesztő aktiv.

*

* ktorol.prs: 5.példa

*

* A-tábla kurzor által kijelölt sorainak törlése

*

- * Kurzor definiálása; ez mutat majd az E-tábla soraira (egyenként).

DECLARE dk CURSOR

FOR SELECT azonosito

FROM dolgozok

WHERE azonosito > "009";

- * Kurzor nyitása, ekkor hajtódik végre a SELECT és a kurzor az E-tábla

- * első sora elé mutat.


```

OPEN dk;
* Feldolgozási ciklus.
DO WHILE .T.                                && Elvileg végtelen ciklus.
    * A kurzor átállítása az E-tábla következő sorára, a sor értékeinek
    * átvitele változókba.
    FETCH dk INTO wazonosito;
    IF SQLCODE # 0
        EXIT                                && Ha az E-tábla végére értünk, akkor vége a
    ENDIF                                    && ciklusnak.
    * Az A-tábla sorának törlése. Azt a sort, amelyet törölni kell,
    * az E-táblának azon sorértéke határozza meg, amelyre a kurzor mutat.
    DELETE
    FROM dolgozok
    WHERE CURRENT OF dk;
ENDDO
    * Kurzor lezárása.
CLOSE dk;
*
* Program vége.
*****
SQL. DO ktorol                                && Program futtatása.

```

A program lefutásának eredményeként a dolgozok A-tábla tartalma:

AZONOSITO	NEV	FOOKOD	OKOD	ALAPFIZ	BEDATUM	NEMKOD
005	O Ibolya	03	02	9000	78.07.08	.T.
007	Szabó Dénes	01	01	22000	54.03.16	.F.
009	Jó Péter	03	01	19000	69.04.07	.F.
008	Kós János	01	01	20000	81.07.12	.F.

Az előzőekben lényegében teljességgel áttekintettük a STANDARD SQL lehetőségeit. Remélhetőleg sikerült érzékeltetnünk, hogy az SQL használata a hagyományos programozási szemlélettől teljesen eltérő szemléletet igényel, és ez nem más, mint a táblákban való gondolkodás szemlélete. Nem könnyű átállni erre, viszont a relációs adatmodellre alapozott alkalmazásfejlesztő rendszerek várható térhódítása előbb-utóbb szükségessé teszi az SQL-re támaszkodó adatbáziskezelő rendszerek megismerését.

Az SQL-re alapozott rendszerek megoldják mind az adatvédelem, mind pedig a hálózati alkalmazás problémáját. Adatvédelmi szempontból a STANDARD SQL-nek is vannak parancsai, ezek konkrétan a fejlesztői rendszerhez igazodnak. A hálózati alkalmazás problémáját általában az alkalmazásfejlesztői rendszer és nem az SQL kezeli.

4. FEJEZET

A HÁLÓS ADATBÁZISOK

4.1. A CODASYL DBTG javaslat

A hálós adatbázisok a hálós adatmodellre támaszkodnak. A hálós adatmodell egy munkacsoport (Data Base Task Group, DBTG) javaslatán alapszik. E munkacsoport 1971-ben tartott konferenciája eredményeként keletkezett jelentés később *CODASYL-javaslat* (Conference on Data System Languages) néven terjedt el.

A CODASYL-javaslat alapelveket tartalmaz a hálós adatmodellekre, illetve adatbázisokra vonatkozóan. Ezek az alapelvek a következők:

- a) Az adatbázisok alapja olyan redundanciamentes adatszerkezet legyen, amelyet több alkalmazó használhat egyszerre.
- b) Az adatok lekérdezése, módosítása egyidejűleg több programból is elvégezhető legyen.
- c) Az adatbázisra vagy annak részeire több keresési stratégiát lehessen megadni.
- d) Az adatbázist kezelő rendszer biztosítson védelmet az illetéktelen hozzáférésekkel szemben, és képes legyen az adatokhoz való egyidejű hozzáfordulásokat kezelni.
- e) Az adatok fizikai elhelyezése központilag legyen megoldható.
- f) Az adatstruktúra hálós kapcsolatokból álljon.
- g) A felhasználó a struktúrában meglévő kapcsolatok karbantartási eljárásainak ismerete nélkül kezelhesse az adatokat.
- h) A programok és az adatstruktúra között ne legyen nagy a függőség (az adatstruktúra bizonyos változtatásai miatt ne kelljen a programokat módosítani).
- i) Az adatbázis úgy legyen leírható, hogy bármilyen programnyelvből elérhető legyen.
- j) Legyen lehetőség arra, hogy az adatbázisban lévő adatok a program számára külön-külön leírhatók legyenek.

A CODASYL-javaslat jelentős szerepet töltött be a hálós adatmodellek kialakulásában. Az azóta létrehozott hálós adatbázisok, illetve az ezeket kezelő nyelvek teljesítik a javaslatban szereplő alapelveket.

4.2. A hálós adatmodell alapelemei

4.2.1. A rekordtípus

A rekordtípust kétféle módon is definiálhatjuk. Az egyik meghatározás szerint a *rekordtípus adattételek névvel ellátott együttese*. A rekordtípus tehát összetartozó adattételekből áll. A másik meghatározás szerint a *rekordtípus a hálós adatmodell legkisebb címezhető egysége*. Ez azt jelenti, hogy a hálós adatmodellben a rekordtípusnál mélyebben nem tudunk az adatokhoz hozzáférni, vagyis csak a rekordtípust tudjuk elérni, a rekordtípus által összefogott adattételeket nem.

A rekordtípusra példa a 4.1. ábrán szereplő adatszerkezet. Ez a rekordtípus a VEVŐNÉV, VEVŐKÓD, VEVŐCÍM adattételekből áll. Ezeket együtt a VEVŐ nevű rekordtípusba fogtuk össze.

VEVŐ

VEVŐNÉV	VEVŐKÓD	VEVŐCÍM
---------	---------	---------

4.1. ábra

Minden rekordtípus értékek egy halmazát reprezentálja. Ezeket az értékeket a rekordtípus *előfordulásainak* nevezzük. Az előbbi rekordtípus előfordulásai lehetnek például:

SZÁMALK, SZT-12, Budapest, Szakasits út 68.

VIDEOTON, VT-52, Székesfehérvár.

A rekordtípus előfordulásai tehát az adattételek konkrét értékeiből állnak össze. (Az első előfordulásnál a VEVŐNÉV adattétel értéke SZÁMALK, a VEVŐKÓD értéke SZT-12, a VEVŐCÍM értéke pedig Budapest, Szakasits út 68.) Ha egy rekordtípus előfordulásait egymás alá írjuk, és az így keletkező oszlopok fölött megadjuk az adattételek nevét, akkor voltaképpen egy relációt kapunk. Így azt is mondhatjuk, hogy a rekordtípus nem más, mint egy *reláció* (a hálós és a relációs adatmodellek között ez az egyetlen közös vonás). A rekordtípust úgy is tekinthetjük, hogy nem más, mint az *egyed típus* hálós adatmodellbeli képe, megfelelője. Ebben a fölfogásban az adattételek voltaképpen *tulajdonságtípusok*.

A rekordtípust a továbbiakban téglalappal jelöljük.

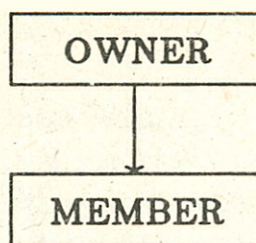
Megjegyzés: a rekordtípust a hálós adatmodell *változójának* tekinthetjük, amelynek azonosítója a rekordtípus neve, értékei pedig a különféle előfordulások.

4.2.2. A settípus

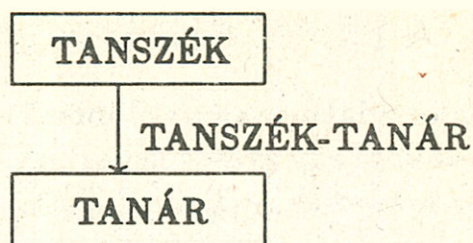
A settípus két rekordtípus, a köztük lévő kapcsolattal együtt. A kapcsolat egy egylépcsős hierarchia, amelyben a fölül álló rekordtípust tulajdonosnak (owner), az alul lévőt pedig tagnak (member) nevezzük. Azt, hogy két rekordtípus kapcsolatában melyiket tekintjük tulajdonosnak, és melyiket tagnak, az adatok közötti tartalmi összefüggés határozza meg.

A settípust a 4.2. ábrán látható módon ábrázoljuk. A két rekordtípust összekötő él fejezi ki a két rekordtípus közötti kapcsolatot. A settípus, mint láttuk, nem egyszerűen két rekordtípus, hanem egy új objektum, ami kifejezi az őt alkotó rekordtípusok közötti kapcsolatot is, ezért *névvel* látjuk el.

Ugyanúgy, mint a rekordtípus, a settípus is értékek halmazát foglalja össze; a settípus egy konkrét értékét a *settípus előfordulásának* nevezzük. Nyilvánvaló, hogy ez az előfordulás a setet alkotó két rekordtípus előfordulásából áll, de több azoknál, hiszen a settípus kifejezi a két rekordtípus közti kapcsolatot is.

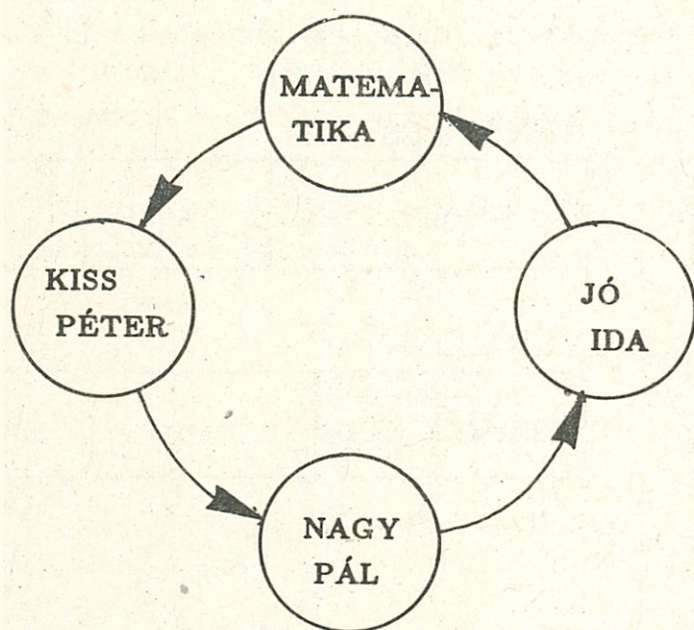


4.2. ábra

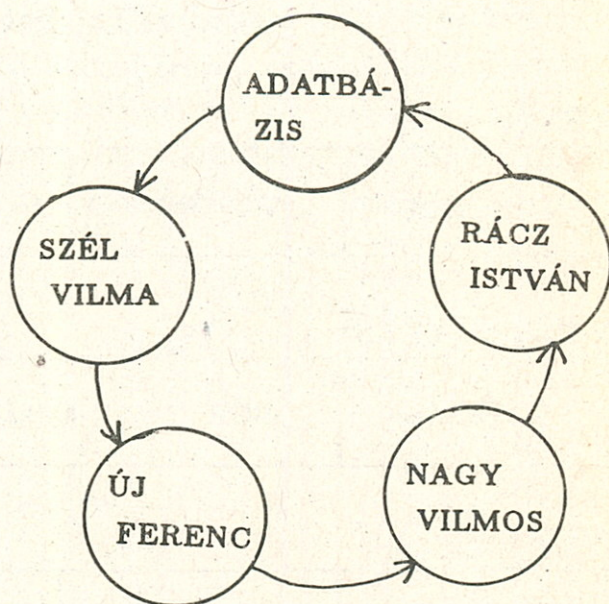


4.3. ábra

Legyen például adva a 4.3. ábrán látható settípus. Ebben a settípusban a tulajdonos a TANSZÉK, a tag pedig TANÁR rekordtípus. A settípus által kifejezett kapcsolat itt azt jelenti, hogy egy tanszéken kik a tanárok, vagyis hogy egy adott tanszéknek kik a tagjai. Ennek a settípusnak egy konkrét értékét (egy előfordulását) úgy adhatjuk meg, hogy megadunk egy konkrét tanszékot, és felsoroljuk mellé az ott dolgozó tanárokat. A szemléletesség kedvéért célszerű ezt a 4.4. ábrán látható módon gyűrűs szerkezetben ábrázolni (a későbbiekben látjuk majd, hogy ez a szerkezet a settípus által kifejezett kapcsolat egy fontos aspektusát is kifejezi). A továbbiakban egy settípus-előforduláson mindig egy ilyen gyűrűs szerkezetet értünk, amelyben a felső körben mindig az owner szerepel, a tagokat pedig nyilakkal kötjük az ownerhez. A TANSZÉK-TANÁR settípus egy másik előfordulását a 4.5. ábrán láthatjuk.



4.4. ábra

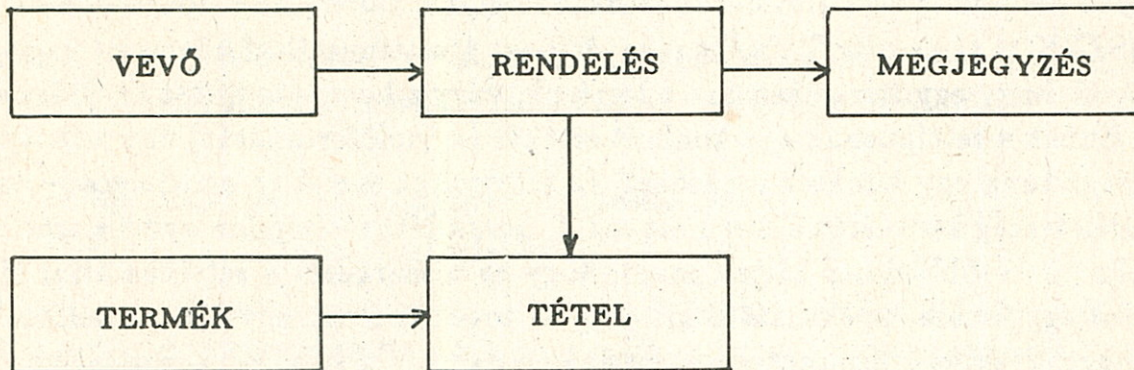


4.5. ábra

Megjegyzés: Mint említettük, a settípust is névvel látjuk el. Ez a körülmény lehetővé teszi, hogy egy másik definícióját is megadjuk a settípusnak. Eszerint egy settípus nem más, mint a hálós adatmodellnek egy, a rekordtípushoz képest magasabban címezhető egysége. A magasabban címezhetőség azt jelenti, hogy a settípus elérése után hozzá tudunk férni annak részeihez, a rekordtípusokhoz is, amelyeknek szintén van nevük, címük.

4.2.3. A hálós adatmodell

Egy hálós adatmodell az előbbiekben definiált két objektumból épül föl, mégpedig úgy, hogy ezekből egy gráfot képezünk. A hálós adatmodell tehát egy gráf, amelynek csomópontjai rekordtípusok, az élek pedig settípusokat jelentenek. A gráf a hálós adatmodell sémája. Szemléletesen szólva egy hálós adatmodell szerint készülő adatbázist megadni annyit jelent, mint fölrajzolni a gráfját, vagyis megadni a sémáját. A 4.6. ábrán egy hálós adatbázis látható.



4.6. ábra

A továbbiakban a hálós adatbázisokkal kapcsolatos ismereteket ezen a példán illusztráljuk, ezért megadjuk az adatbázisban szereplő rekordtípusok adattételeit is.

VEVŐ

VEVŐKÓD	VEVŐNÉV	CÍM
---------	---------	-----

RENDELÉS

RENDELÉS-SZÁM	DÁTUM	...
---------------	-------	-----

MEGJEGYZÉS

SORSZÁM	SOR	MDÁTUM
---------	-----	--------

TERMÉK

TERMÉK-KÓD	TERMNÉV	EÁR
------------	---------	-----

TÉTEL

TÉTELSZÁM	MENNYISÉG	INDEX
-----------	-----------	-------

4.7. ábra

Nagyon röviden ugyan, de az előbbiekben elmondtuk a hálós adatmodell lényegét. Azokat az adatbázisokat, melyek szerkezete hálós adatmodellel írható le, *hálós adatbázisoknak* hívjuk.

4.3. Az adatbázis elemeinek megadási módja

4.3.1. A settípus megadási módja

Egy hálós adatbázis megszerkesztésénél nyilvánvaló kérdés, vajon hogyan tudjuk megadni az adatbáziskezelő rendszer számára a két rekordtípus közötti kapcsolatot, más szóval hogyan tudjuk leírni a két rekordtípust összekötő nyilat. Honnan tudják a rekordtípus-előfordulások, hogy melyik settípusban vesznek részt. A válasz: a settípusban szereplő rekordtípus-előfordulások közötti logikai kapcsolatokat mutatókkal (ponterekkel) adjuk meg. Szemléletesen szólva a settípus két rekordtípusa közötti nyilat a rekordtípusok előfordulásaira vetítjük rá, vagyis ezt a nagy nyilat az előfordulások közötti sok kis nyílra bontjuk szét.

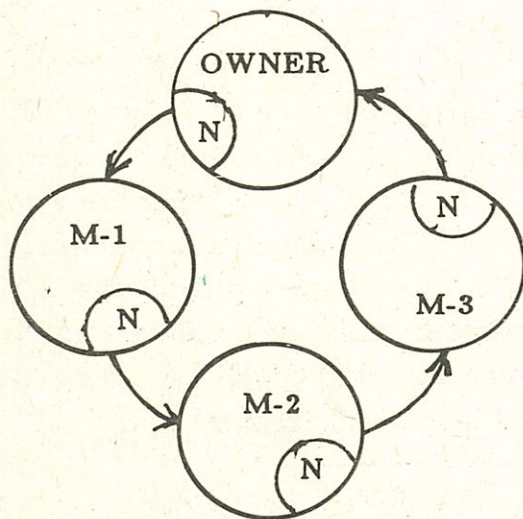
Minden settípusban háromféle pointer szerepelhet. Ez azt jelenti, hogy a settípusban szereplő rekordtípus-előfordulások közötti logikai kapcsolatot háromféleképpen fejezhetjük ki.

Azt, hogy egy settípusban milyen pointerek legyenek, mi döntjük el, amikor az adatbázist megtervezzük.

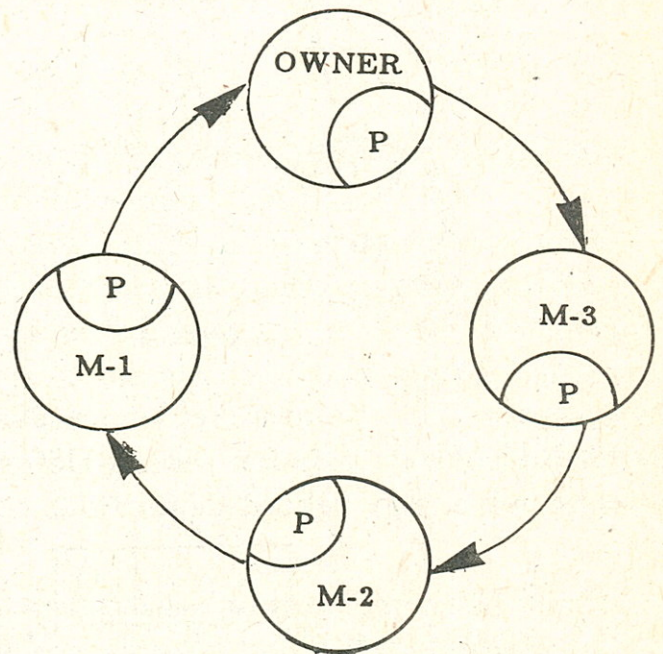
a) A NEXT pointer

Ilyen pointert minden settípusnak tartalmaznia kell. Ez a pointer azt mutatja, hogy melyik rekord-előfordulást melyik követi a setben (logikailag). Ebben az esetben a set-előfordulás alakzata a 4.8. ábrán látható szerkezetű lesz.

Tehát: a tulajdonosból pointer mutat az első memberre, az első memberből a második memberre s.í.t., az utolsó memberből pointer mutat az ownerre. A tagok a NEXT pointerekkel vannak „ráfűzve” a tulajdonosra. Kereséskor ezt jól ki tudjuk használni: ha az ownert megtaláltuk, elővehetjük az ő (logikailag) első memberét, majd az ezután következőt stb.



4.8. ábra



4.9. ábra

b) A PRIOR pointer

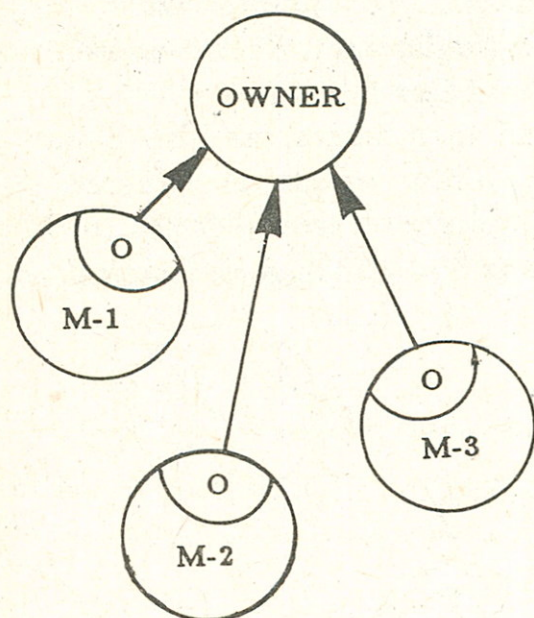
Ha PRIOR pointer is van a settípusban, akkor a settípus-előfordulás szerkezete a 4.9. ábrán látható alakzatú lesz. Ez azt jelenti, hogy a set minden tagjából pointer mutat az előtte lévő tagra, az utolsó tagtól az ownerre, az ownertől az utolsó tagra.

Amennyiben az adatbázis megtervezésénél úgy döntünk, hogy egy settípusban legyen PRIOR pointer, akkor az előfordulások keresésekor ki tudunk adni olyan utasítást, hogy hozza elő a rendszer egy már korábban megkeresett előfordulás előtti előfordulást.

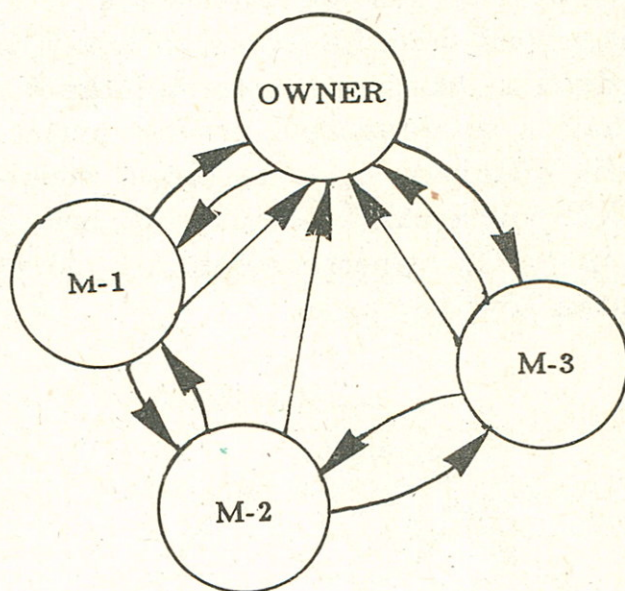
c) Az OWNER pointer

Megjelölhetjük egy settípus minden memberében azt is, hogy melyik rekordtípus az ő ownere. Ebben az esetben a settípus-előfordulás szerkezete a 4.10. ábrán látható alakzatú lesz. Amennyiben egy olyan settípusban keresünk vissza az adatbázisban, amelyben van owner pointer, akkor akármelyik membernél is vagyunk (egy korábbi keresési művelet eredményeként), meg tudjuk keresni az ownerét.

Amennyiben mind a három pointer szerepel a settípusban (ezt mi döntjük el azzal, hogy a NEXT-nek mindig szerepelnie kell), akkor a settípus-előfordulások alakzata a 4.11. ábrán látható szerkezetű lesz.

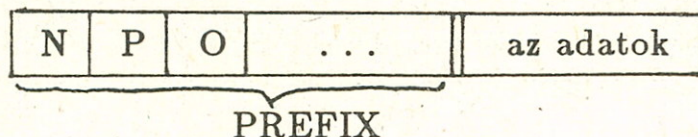


4.10. ábra



4.11. ábra

Megjegyzés: A pointereket a rendszer automatikusan helyezi el a rekord-előfordulásokban. Minden rekord-előfordulásnak van egy ún. PREFIX része; ez a pointernek a helye. Egy rekord-előfordulás tehát a következő szerkezetű:



A settípusok tehát igazi *listaszerkezetet* alkotnak; ennél fogva egy hálós adatbázisban a rekord-előfordulások egy lista elemei (a listában minden elemet pointernek kötnek egymáshoz). Az adatbázisban a keresés a listán való útbejárást jelent.

4.3.2. Az adatok fizikai elhelyezése

Az előbbiekben ismertettük, hogyan „valósulnak” meg az adatbázisban a logikai kapcsolatok. Ezt az adatbázis tervezői határozzák meg.

Az adatok (rekord-előfordulások) fizikai elhelyezésének módját is mi adjuk meg az adatbázis megtervezésénél.

a) A *CALC* elhelyezési mód

A *CALC* elhelyezési mód a rekordtípus-előfordulások fizikai helyét az adatbázisban egy eljárással számítja ki (*CALC* a *CALCULATION* rövidítése). Nekünk kell megmondanunk, hogy a rekordtípus melyik adattételéből számítsa ki a helyet.

Tegyük fel például, hogy a 4.7. ábrán látható szerkezetű *VEVŐ* rekordtípus előfordulásait akarjuk *CALC*-kel elhelyezni. Dönthetünk úgy, hogy minden *VEVŐ*-nél a *VEVŐKÓD* adattételből számítsa ki a rendszer egy előfordulás helyét. Minden vevőnek (*VEVŐ*-előfordulásnak) van egy *VEVŐKÓD*-ja, ezt veszi a *CALC* operandusként, belőle kiszámít egy egész számot; ez lesz az adott vevő helye (címe) az adatbázisban.

Például, ha a *SZÁMALK*-ot akarjuk elhelyezni, azaz a *VEVŐ* rekordtípus

SZÁMALK SZT-12 Budapest Szakasits Á. 68.

előfordulásának a helyét számítja ki a *CALC* eljárás, akkor az *SZT-12* vevőkódot veszi operandusnak (ebből készít egy egész számot).

b) *VIA* elhelyezési mód

A *VIA* elhelyezési mód csak emberek fizikai helyének kijelölésére való. Amennyiben egy rekordtípus előfordulásait *VIA*-val akarjuk elhelyeztetni, akkor az adatbázis tervezésekor ezt közöljük a rendszerrel. Ez az „intézkedésünk” azt jelenti, hogy a member-előfordulásokat az owner rekordhoz fizikailag a legközelebb helyezi el a rendszer.

c) *DIRECT* elhelyezési mód

A *DIRECT* elhelyezési mód esetén mi adjuk meg a rekord-előfordulás fizikai helyének a címét. (Csak nagyon kivételes esetben érdemes használni.)

Megjegyzés: A fenti módon elhelyezett rekord-előfordulások számára névvel ellátott speciális területeket, ún. *AREA*-kat kell kijelölnünk az adatbázisban. Egy *AREA* folyamatosan sorszámozott lapokból áll. (A lap egy tárolási egység; a rekord-előfordulások ilyen lapokra kerülnek rá.) Azt, hogy egy rekordtípus előfordulásai melyik területre kerüljenek, az adatbázis definiálásakor mi mondjuk meg.

4.3.3. A set rendje

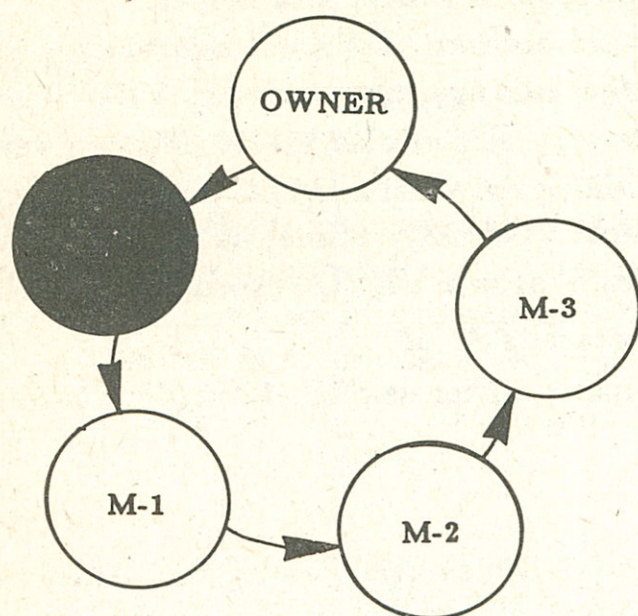
Amikor az adatbázist létrehozuk (definiáljuk), meg kell mondani, hogy az új tagot hova „szúrja” be, hogyan helyezze el (logikailag) a settípuson belül. Ennek ötféle módja van, mi itt hármat ismertetünk.

a) A *FIRST*-mód

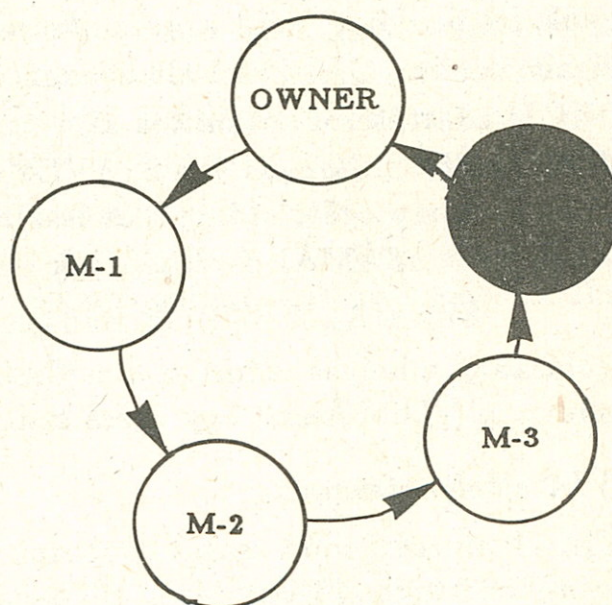
Ha az adatbázis definiálásánál úgy intézkedünk, hogy az új member-előfordulást *FIRST* módban szúrja be (a többi, már létező közé), akkor ezt a tulajdonos és a (logikailag) első member közé teszi (úgy, mint ahogy a 4.12. ábrán láthatjuk, ahol a sötétített kör az új member-előfordulás).

b) *LAST*-mód

LAST beszúrási mód esetén a helyzet a 4.13. ábrán látható (azaz a tulajdonos és az utolsó member közé szúrja be az új member-előfordulást).



4.12. ábra



4.13. ábra

c) *SORTED*-mód

Amennyiben *SORTED*-módot írunk be az adatbázis-definíció valamely settípusának leírásánál, akkor az új member-előfordulásokat abban a setben rendezetten helyezi el; mégpedig a rekordtípus valamely adattétele szerinti sorrendben. Azt, hogy melyik legyen ez az adattétel, mi adjuk meg. (Ez az adattétel a *SORTED* kulcsa, rendezési kulcs.)

Ha például a *VEVŐ-RENDELÉS* settípusban a *RENDELÉS* tagokat *SORTED*-dal akarjuk beszúrni a settípusba, akkor dönthetünk úgy, hogy a *RENDELÉS* rekordtípus rendezési kulcsa a *HATIDŐ* (határidő) adattétel legyen. vskip-3pt

4.4. Az adatbázis sémájának grafikus leírása

Az adatbázis szerkezetének (sémájának) grafikus leírására a hálós adatmodellben egy „szabványt” javasolnak. E szabvány alapján a 4.6. ábrán megadott adatbázist például a 4.14. ábrán látható módon írhatjuk le.

4.4.1. A rekordtípusok ábrázolása

A sémában a rekordtípusokat ábrázoló téglalapok soraiban a következő információk szerepelnek:

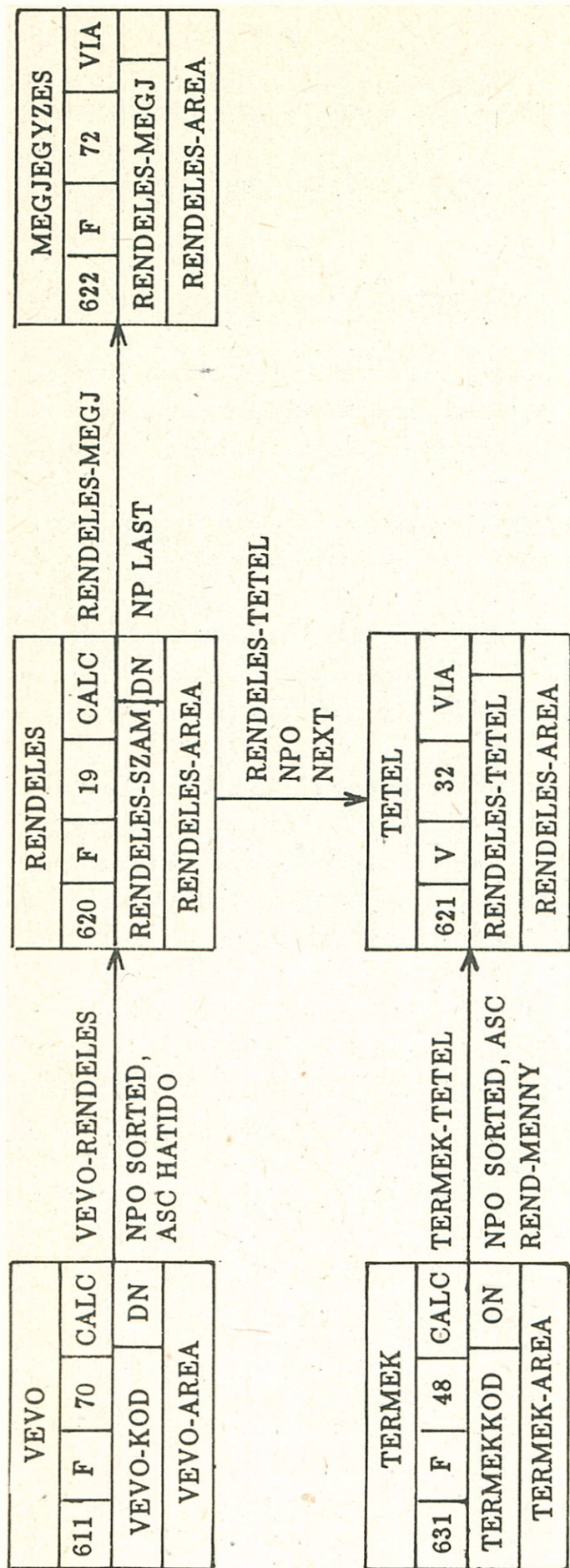
1. Az első sorban a rekordtípus nevét adjuk meg. (A gráf bal felső rekordtípusának neve: VEVO.)
2. A második sorban négy paramétert adunk meg.
 - a) Az első helyen a rekordtípus egy azonosítószáma szerepel (egy egész szám, mi adjuk meg; a VEVO-nél ez 611).
 - b) A második kis téglalapban azt adjuk meg, hogy a rekordtípus-előfordulások hossza milyen legyen: ide V betűt írunk, ha változó hosszúságú rekordot akarunk definiálni, F-et, ha fix hosszúságút. (A VEVO-nél fix hosszúságú mindegyik előfordulás.)
 - c) A harmadik helyre az előfordulások hosszúságát írjuk be (byte-okban mérve; változó hosszúság esetén ez a maximális hosszúságot jelenti). A VEVO rekordtípusban – mint látjuk – a hosszúság: 70 byte.
 - d) A második sor negyedik helyére az elhelyezési módot írjuk (mint már tudjuk, ez lehet CALC, VIA vagy DIRECT). A VEVO-nél úgy döntöttünk, hogy az elhelyezési mód CALC, de mint a sémán látjuk, a MEGJEGYZES (és a TETEL) rekordtípusoknál az elhelyezési mód VIA.
3. A harmadik sorban az első helyre CALC-kel történő elhelyezés esetén a CALC-nek a kulcsát (annak az adattételnek a nevét, amelyből a CALC a címet kiszámítja) kell írunk, VIA esetén pedig annak a settípusnak a nevét, amelyben a membert az ownerhez legközelebb akarjuk elhelyezni. Sémánkon a CALC-kel elhelyezett VEVO CALC-kulcsa a VEVOKOD, a MEGJEGYZES rekordtípusnál pedig a settípus: RENDELES-MEGJ.

A harmadik sor második helyén arról intézkedünk, hogy amennyiben a CALC-eljárás két különböző rekordtípusnak ugyanazt a helyet adná meg (a számítás eredményeként ez jönne ki), akkor ezt elfogadjuk-e (ezt úgy jelöljük, hogy DUPLICATION ALLOWED=DA), és akkor a rendszer a második (iker) rekord-előfordulást egy túlsordulási területre teszi, vagy ezt nem engedjük meg (ekkor DUPLICATION NOT ALLOWED=DN-t írunk). Mint a sémán látjuk, a VEVO-nél nem engedünk meg duplikációt.
4. Végül a negyedik sorban azt adjuk meg, hogy a rekordtípus előfordulásait melyik AREA-ra akarjuk elhelyezni. A VEVO-nél a VEVO-AREA nevű AREA-t jelöltük ki.

4.4.2. A settípusok ábrázolása

A settípusok grafikus megadásánál az élre (a nyílra) írjuk az információkat (a settípus paramétereit) mégpedig:

- a) a settípus nevét (a bal felső settípusban: VEVO-RENDELES);
- b) a pointereket (a VEVO-RENDELES settípusban mind a NEXT, mind a PRIOR, mind az OWNER pointert megadjuk, ezért NPO került az ábrába, a RENDELES-MEGJ settípusban viszont csak NP-t adtunk meg);



4.14. ábra

- c) a set rendjét (a VEVO-RENDELES settípusban a set rendje SORTED, növekvőleg (ASC) és a rendezés a HATIDO adattétel szerint történik, a RENDELES-MEGJ settípusban LAST a set rendje.

Az adatbázis grafikus ábrázolásával voltaképpen teljesen megadtuk az adatbázist; egy konkrét adatbáziskezelő nyelven leírni gyakorlatilag annyit jelent, mint az adott nyelv „szavaival” (utasításai-val) „lefordítani” a táblát a gép számára. vskip-2pt

4.5. Az IDMS adatbáziskezelő rendszer

A hálós adatbáziskezelő rendszerek közül a legelterjedtebb az IBM gépekre készített IDMS (Integrated Database Management System), amelyet a hetvenes évek közepén a Cullinet szoftverház hozott létre.

Az IDMS nagyon nagy rendszer. Több alrendszerből áll. Mi itt csak a rendszer központi magjával foglalkozunk. A központi mag is több alnyelvet tartalmaz. Ezek közül röviden kettőt tekintünk át: az *adatdefiniáló* és az *adatlekérdező* nyelvet. (Áttekintésünk nagyon vázlatos; célunk egyszerűen a lényeg bemutatása.)

Mint könyvünk elején is említettük, az IDMS nem valóságos programnyelv, hanem ún. host-nyelv (befogadó típusú nyelv); csak igazi programnyelvvvel együtt (pl. Cobol, PL-1 stb.) használható.

4.5.1. Az adatbázis definiálása (a DDL nyelv)

Az adatbázis szerkezetének (sémájának) leírása az IDMS-DDL nyelven (Data Definition Language) történik.

A séma leírásának módját a 4.14. ábrán szereplő adatbázison mutatjuk be. A séma leírása több részből áll; ezeket a részeket olyan sorrendben írjuk le a továbbiakban, ahogy az a DDL-ben elő van írva. Az egyes részeket elválasztó címeket és a néhány soros magyarázatot csak a tagolás céljából, az „olvashatóság” kedvéért tettük be.

Könyvünkben a DDL utasításait nagybetűkkel, az utasítások mellé írt paramétereket dőlt nagybetűkkel írjuk. Az „álló nagybetűs” részeket olyan szabványosan és sokszor ismételve kell az IDMS-ben leírni, hogy a séma leírásához nyomtatványt is lehetne készíteni. A programrészek mellé az ott szereplő szavakra kis szótárt adunk.

A sémaleírás

a) A sémafej

Ebben a részben elnevezzük a sémát, továbbá megadhatjuk ki mikor és hol készítette, ill. megjegyzéseket tehetünk.

SCHEMA DESCRIPTION.
SCHEMA NAME IS MINTA.

AUTHOR KISS PÁL.
DATE 05.02.90.

INSTALLATION. SZÁMALK.

REMARKS. EZ A MINTA ADATBÁZIS SÉMÁJA.

(SCHEMA=séma, DESCRIPTION=leírás)
(NAME=név, IS=van)

(AUTHOR=szerző)
(DATE=dátum)

(INSTALLATION=üzembeállítás)

(REMARKS=megjegyzések)

b) *Az area-leírás*

Ebben a részben az AREA-k neveit és a benne szereplő lapok kezdő- és végsorszámát adjuk meg.

AREA DESCRIPTION.

AREA NAME IS VEVO-AREA.

RANGE IS 2001 THRU 2090. (RANGE=tartomány, THRU=keresztül [tól-ig])

AREA NAME IS...

(A többi area leírása)

c) *A rekordleírás*

A rekordleírás lényegében nem jelent mást, mint azt, hogy a 4.14. ábrán látható rekordokról szóló információt ide (szabványos alakban) átmásoljuk.

RECORD DESCRIPTION.

RECORD NAME IS VEVŐ.

RECORD ID IS 611.

LOCATION MODE IS CALC

USING VEVOKOD.

DUPLICATES ARE NOT ALLOWED

WITHIN VEVO-AREA AREA.

03 VEVOKOD PIC X(10)

03 VEVONEV PIC X(20)

03 CIM PIC X(40)

:

(LOCATION MODE = elhelyezési mód

(USING = használva)

(DUPLICATES ARE NOT ALLOWED =
duplikáció nincs megengedve)

(WITHIN = -ban, - ben)

A VEVO rekord CALC tárolási módú, a CALC kulcsa a VEVOKOD, és nem engedjük meg az azonos CALC kulcsú rekordokat. A rekordot a VEVO-AREA nevű AREA-ban helyezzük el. A rekordleírás végén a rekord adattételeit adjuk meg. Ez az egyetlen információ, ami nem szerepel az adatbázis szabványos rajzán (a 4.14. ábrán), hanem külön írjuk le. A fenti rekordleírásban szereplő VEVO rekordtípus szerkezete:

VEVO

VEVOKOD	VEVONEV	CIM
---------	---------	-----

A rekordtípus szerkezetét az IDMS-ben ugyanúgy írjuk le, mint a Cobolban. Az adattételek után szereplő PIC-kel (PICTURE, ejtsd: pikcsör) az adattétel típusát és hosszát adjuk meg. PIC X(10) azt jelenti, hogy a VEVOKOD karakteres típusú és 10 byte hosszú.

A fentihez hasonló módon definiáljuk az adatbázisban szereplő összes rekordtípust, azaz szabványos alakban átmásoljuk ide a 4.14. ábrán látható rekordokhoz tartozó

információkat. Ott, ahol az elhelyezési mód VIA (példánkban a MEGJEGYZES és TETEL rekordtípusoknál van így),

LOCATION MODE IS CALC USING...

helyett a

LOCATION MODE IS VIA *setttípusnév* SET
utasítást írjuk.

A példánkban szereplő MEGJEGYZES rekordtípusnál tehát ezt írjuk:
LOCATION MODE IS VIA RENDELES-MEGJ SET.

c) *A set-leírás*

A setek leírása is a grafikus sémában szereplő információknak a DDL „mondataival” való leírását jelenti. Adjuk meg a VEVO-RENDELES setet.

SET DESCRIPTION.

SET NAME IS *VEVO-RENDELES*

ORDER IS *SORTED*.

(ORDER=rend)

OWNER IS *VEVO*

NEXT DBKEY POSITION IS *1*

PRIOR DBKEY POSITION IS *2*

MEMBER IS *RENDELES*

NEXT DBKEY POSITION IS *1*

PRIOR DBKEY POSITION IS *2*

OWNER DBKEY POSITION IS *3*

ASCENDING KEY IS *HATIDO*

A VEVO-RENDELES settípusban az új member elhelyezése rendezetten történik, ezért a 3. sorban ORDER IS SORTED szerepel. Azt, hogy növekvőleg akarjuk elhelyezni, és hogy a rendezés a HATIDO szerint legyen, az utolsó sorba kell beírni.

Az ownernél is, a membernél is meg kell adnunk, hogy a NEXT, PRIOR, OWNER mutatók (pointerek) a Prefix melyik pozíciójában helyezkedjenek el. Az 5. sorban szereplő

NEXT DBKEY POSITION IS *1*

azt jelenti, hogy a VEVO rekordtípus (az owner) NEXT pointere a prefix 1. pozíciójában lesz; a

PRIOR DBKEY POSITION IS *2*

pedig azt, hogy a PRIOR pointer helye a 2. pozíció. Ugyanezt jelenti a membernél szereplő első két sor is, míg a harmadik sor,

OWNER DBKEY POSITION IS *3*

az OWNER pointer helyét adja meg a prefixben.

A VEVO-RENDELES settípus után a többi settípus definíciója következik. Ezek közül a RENDELES-MEGJ és a RENDELES-TETEL settípus a set rendjében különbözik, előbbinél a második sorba ORDER IS LAST, az utóbbinál pedig ORDER IS NEXT utasításokat kell írunk (mert a séma grafikus leírásában ez szerepel, vagyis amikor felrajzoltuk „struktúrát”, így döntöttünk).

Az alséma leírása

Az alsémával egy felhasználói „nézetet” emelünk ki az adatbázisból. A felhasználói programokat mindig valamilyen alsémára alapozva írjuk. Ezt úgy is felfoghatjuk, hogy minden felhasználói program egy „saját” adatbázisból végzi a feldolgozást, ami nem azt jelenti, hogy fizikailag is létrejön a saját adatbázis (az alséma is csak egy logikai séma).

Az alséma leírása szerkezetileg megegyezik a séma leírásával. Szerencsére nem kell az alsémába „kiemelt” area-, rekord-, setleírásokat szóról szóra bemásolni, elég egy COPY utasítást írni, aminek a hatására átmásolódnak a sémából a megfelelő elemek.

Készítsünk a MINTA sémából egy PROBA nevű alsémát.

SUBSCHEMA IDENTIFICATION DIVISION. (= alséma, azonosító, rész)

SUBSCHEMA NAME IS *PROBA*.

OF SCHEMA NAME *MINTA*.

AUTHOR *NAGY PAL*.

DATE *28.11.90*.

INSTALLATION *SZAMALK*.

REMARKS. *MINTA ADATBAZIS PROBA ALSEMAJA*.

SUBSCHEMA DATA DIVISION (DATA = adat)

AREA SECTION (SECTION = szekció)

COPY *VEVO-AREA* AREA (COPY = másolat)

⋮

RECORD SECTION

COPY *VEVO* RECORD

⋮

SET SECTION

COPY *VEVO-RENDELES* SET

⋮

A pontok helyén további COPY-k állnak; ide azokat a további objektumokat írjuk be, amiket az alsémába át akarunk hozni. Miután az alsémát definiáltuk, használhatjuk programjainkban. Mint említettük, az alséma is a séma által meghatározott adatbázisból veszi az adatokat, tehát minden adat egyszeresen van tárolva. Ilymódon a felhasználói programok az alsémákon keresztül férnek hozzá a fizikailag tárolt adatokhoz.

4.5.2. A visszakereső és módosító utasítások

Az előzőekben a DDL utasításait ismertettük. Ezek nem „igazán” végrehajtható utasítások. Azok az utasítások, amelyekkel az adatbázisban lévő adatokat ténylegesen kezelni is tudjuk, az IDMS DML alnyelvében (Data Manipulation Language) vannak.

Mivel célunk csupán rövid áttekintést adni az IDMS nyelv elemeiről, ezért a DML-ből is csak néhány utasítást ismertetünk.

A DML-ben nincsen sok utasítás. Soroljuk is fel őket mindjárt itt az elején (röviden megadva egyúttal a jelentésüket is).

1. FIND: megkeres egy rekordot (a rekord-előfordulást röviden rekordnak is hívjuk) az adatbázisban;
2. OBTAIN: megkeres és beolvas egy rekordot;
3. GET: beolvas egy megkeresett rekordot. Megjegyezzük, hogy a FIND és OBTAIN utasítás paraméterezése teljesen azonos. Az azonos paraméterezésű FIND és OBTAIN utasítás között az a különbség, hogy az OBTAIN beolvassa a rekordot, a FIND nem. A GET a FIND utasítással megkeresett rekord beolvasására szolgál (tehát $OBTAIN = FIND + GET$).
4. IF: adatbázis állapotok tesztelése;
5. STORE: rekord tárolása;
6. MODIFY: rekord módosítása;
7. ERASE: rekord törlése;
8. CONNECT: rekord bekapcsolása setbe;
9. DISCONNECT: rekord kikapcsolása setből.

A felsorolt utasítások voltaképpen utasításcsoportok. Mindegyiknek több változata van; ezeket lényegében különböző utasításoknak tekinthetjük. Mi itt csak az OBTAIN/FIND utasítás néhány változatát ismertetjük (példákon keresztül). Előtte azonban két fontos fogalommal kell megismerkednünk.

A kurrenség fogalma

A DML utasításai rekord-előfordulásokkal végeznek műveleteket. Azt a rekordot, amit egy DML utasítással legutoljára kezeltünk (megkerestük, módosítottuk stb.), *kurrens* előfordulásnak (rekordnak) nevezzük. Aszerint, hogy „honnan nézve” történt ez legutoljára, több eset lehetséges.

1. A *program* kurrens rekordja az, amelyhez a programból utoljára nyúltunk hozzá.
2. Az *area* kurrens rekordja az az előfordulás, amit abban az areában utoljára kezeltünk.
3. Egy *rekordtípus* kurrens rekordja az, amelyet abban a rekordtípusban utoljára kezeltünk.
4. Egy *set*típus kurrens előfordulása az, amelyikhez abban a settípusban utoljára nyúltunk hozzá.

A navigációs út

A hálós adatbázisban egy rekord megkeresése egy út „bejárását” jelenti. Mivel az adatbázis szerkezetét egy gráf írja le, ezért ez az út tényleg egy gráf útja, azaz élek sorozata.

Tegyük fel például, hogy a MINTA adatbázisban meg akarjuk keresni egy konkrét vevő konkrét megrendelésében szereplő konkrét tételt. Ezt úgy tehetjük meg, hogy belépünk az adatbázisba annál a VEVO előfordulásnál, akiről szó van. Ettől a vevőtől él vezet (pointer mutat) a gráfban a vevőhöz tartozó rendelésekre (a RENDELES rekordtípusra). Innen viszont egy él megy a TETEL rekordtípusra, amelyben meg tudjuk találni a konkrét tételt.

Ha volnának olyan utasításaink, amelyekkel be tudunk lépni a gráfba (az adatbázisba), és utána élről élre (csomópontból csomópontba, vagyis rekordtípusról rekordtípusra) tovább tudnánk haladni, akkor meg tudnánk keresni pl. egy konkrét tételt. Egy ilyen utat, amit keresés közben bejárunk, *navigációs útnak*, az ilyen út bejárását lehetővé tevő utasításokat *navigáló utasításoknak* nevezzük. Az alábbiakban röviden ismertetjük azokat az utasításokat, amelyekkel belépni, illetve navigálni lehet az adatbázisban.

Kereső utasítások

1. Az *OBTAIN CALC* utasítás

Ezzel az utasítással olyan rekordtípus előfordulását lehet megkeresni, amit *CALC*-kel helyeztünk el.

Az utasítás formátuma:

OBTAIN CALC rekordtípusnév

Hatása: a *CALC* eljárás kiszámít egy címet, és erről beolvassa a munkaterületre az ezen a címen lévő előfordulást abból a rekordtípusból, amit a *CALC* mögé írtunk.

Tudjuk, a *CALC*-nek meg kell adni egy argumentumot, amiből a címet ki tudja számítani. Amikor a *CALC* eljárást a rekord-előfordulások elhelyezésére használtuk, a *CALC* mellé megadtunk egy kulcsot, mégpedig valamelyik adattételt. Ahhoz, hogy a *CALC* visszakereséskor is működni tudjon, szintén meg kell adni ezt a kulcsot. A megoldást legegyszerűbb példán megmutatni.

Példa: Legyen a feladat a *MINTA* adatbázisban a *VEVO* rekordtípusból megkeresni a *SZAMALK*-ot.

A *VEVO* rekordtípus előfordulásait *CALC*-kel helyeztük el, és a *CALC* argumentuma a *VEVOKOD* adattétel volt.

„Visszefelé” alkalmazva a *CALC*-et, most el kell helyoznünk a *SZAMALK* vevőkódját (*SZT-12*-t) a *CALC VEVOKOD* kulcsába, és erre kiadni az előbbi utasítást.

Az *SZT-12* kód behelyezése a *VEVOKOD*-ba nem az *IDMS* nyelven hanem a behívó nyelven (pl. *Cobol*-ban) történik. Jelöljük ezt az értékadást a következőképpen:

SZT-12 $\Rightarrow$ *VEVOKOD*

(Tehát az *SZT-12* kód bekerül a *VEVOKOD* mezőre, ami a *CALC*-nek a kulcsa.) Ezzel a fenti feladat megoldása az alábbi két sorban írható fel:

SZT-12 $\Rightarrow$ *VEVOKOD*

OBTAIN CALC VEO

Amikor a gép végrehajtja ezt a két utasítást, a *VEVO* rekordtípus

SZAMALK, *SZT-12*, Budapest, Szakasits út 68.

előfordulása megjelenik egy munkamezőben, és a behívó programban fel lehet dolgozni.

2. Az *OBTAIN NEXT* utasítás

Az utasítás formátuma:

OBTAIN NEXT rekordtípus *WITHIN* settípus

Hatása: megkeresi azt az előfordulást a megadott rekordtípusban, amelyre a *NEXT* pointer mutat a settípuson belül. Ezt az utasítást is egy példán mutatjuk be.

Az előbbi kis programrészlettel beléptünk az adatbázisba, mégpedig a SZAMALK előfordulásnál.

Keressük meg most a SZAMALK-hoz tartozó első rendelést. Ezt az alábbi utasítással tehetjük meg:

OBTAIN NEXT RENDELES WITHIN VEVO-RENDELES

Könnyen meggyőződhetünk arról, hogy ez az utasítás valóban a SZAMALK-hoz tartozó első rendelést keresi meg az adatbázisban. Ugyanis a SZAMALK owner a VEVO-RENDELES settípusban. Mint a 4.8. ábrán láthatjuk, az ownerre gyűrűben vannak ráfűzve a tagok, tehát esetünkben a konkrét VEVO előfordulásra a rendelések (a VEVO-RENDELES settípusban). Az ownerből (jelen esetben a SZAMALK-ról) a NEXT pointer az „első” tagra mutat, tehát ez az utasítás éppen a SZAMALK első rendelését állítja elő.

3. Az *OBTAIN-USING* utasítás (keresés rendezett setben)

Az utasítás formátuma:

OBTAIN rekordtípus WITHIN settípus
USING rendezési kulcs

Hatása: az adott settípusban megkeresi azt az első tag-előfordulást, amelyben a rendezési kulcs értéke egy előre adott értékkel egyezik.

Példa: Keressük meg a SZAMALK-nak azt a rendelését, amelynek kielégítését egy adott dátumra (pl.: 20.12.90) ígértük.

Tegyük fel, hogy az előbb mutatott módon (OBTAIN CALC VEVO) a SZAMALK-ot megkerestük; akkor már csak arról kell gondoskodni; hogy az egyébként HATIDO (=határidő) szerint rendezett RENDELES rekordtípusban a rendezési kulcs (HATIDO) vegye fel a fenti dátumot. Programrészletünk a következő lesz:

20.12.90 $\Rightarrow$ HATIDO (a behívó nyelven)
OBTAIN RENDELES WITHIN VEVO-RENDELES
USING HATIDO

A fenti utasítások hatására a 20.12.90 dátum bekerül a HATIDO munkamezőre; ennek alapján az IDMS megkeresi azt a rendelést, aminek a HATIDO adattétele (ami szerint rendezve van) éppen a 20.12.90 dátum.

Ha az adatbázisban a SZAMALK-nak több olyan rendelése van, aminek határideje 20.12.90., akkor az előbbi utasítással az első ilyen kerestük elő; a következő 20.12.90. határidejű rendelést az

OBTAIN NEXT RENDELES WITHIN VEVO-RENDELES

utasítással találhatjuk meg. (Ugyanis settípus kurrens rekordja a 20.12.90 határidejű RENDELES, mivel az előbb ezt kerestük meg; a settípusban ezt kezeltük utoljára.) Ha ezt az utasítást egy ciklusba építjük be, amely figyel, hogy van-e még további 20.12.90 határidejű RENDELES, akkor ily módon a SZAMALK összes megadott határidejű rendelését elő tudjuk hozni.

4. Az OBTAIN OWNER utasítás

Az utasítás formátuma:

OBTAIN OWNER WITHIN settípus

Hatása: megkeresi egy member-előfordulás ownerét.

Példa: Keressük meg egy megadott termékhez tartozó vevőt (azaz egy megadott TERMEK-előfordulás VEVO-előfordulását). Legyen a termékkód: TSZ-52. Mint a 4.14. ábrán látjuk, a TERMEK előfordulásokat CALC-kel kereshetjük meg (mert ezzel helyezzük el), és a CALC kulcsa a TERMEKKOD, ide kell „beállítanunk” a TSZ-52-öt. A program vázlata a következő lehet:

T-52 $\Rightarrow$ TERMEKKOD

OBTAIN CALC TERMEK (megkeresi a T-52-es termékkódú terméket)

OBTAIN NEXT TETEL WITHIN TERMEK-TETEL (a T-52-es kódszámú terméknek – mint ownernek – megkeresi az első memberét, azt, amelyikre ennek a terméknek a NEXT pointere mutat)

OBTAIN OWNER WITHIN RENDELES-TETEL (megkeresi az előbb megtalált tétel ownerjét, vagyis azt a rendelést, aminek ez a tétele)

OBTAIN OWNER WITHIN VEVO-RENDELES (megkeresi az előbb megkeresett rendelés ownerjét, vagyis azt a vevőt, aki ezt a rendelést adta ki; ha több van akkor ezek közül az elsőt)

Ezzel a feladatot megoldottuk. Amennyiben több vevő rendelt ilyen terméket, akkor ciklusba szervezve a fenti utasításokat elő tudjuk hozni mindegyik T-52-es termékkódú terméket rendelő VEVO-előfordulást.

A példa arra is jó volt, hogy egy „hosszabb” navigációs út bejárását mutassuk be az olvasónak.

Tartalomjegyzék

Előszó	5
1. Adatmodellek, adatbázisok	7
1.1. Bevezetés	7
1.2. Adatbáziskezelő rendszerek	8
1.3. Az adatmodellek alapelemei	9
1.4. Az adatmodell és az adatbázisok	11
2. A relációs adatmodell	15
2.1. A relációs adatmodell meghatározása	15
2.2. A relációs adatbázis belső szerkezete	17
2.3. A relációs algebra	32
2.4. Relációs adatbáziskezelő nyelvek	37
3. AZ SQL ADATBÁZISKEZELŐ NYELV	39
3.1. Az SQL adatbázis fogalma és kezelésének parancsai	39
3.2. Adattábla definiálása	41
3.3. Új sorok elhelyezése az adattáblába	43
3.4. Lekérdezés adattáblákból	46
3.5. Táblák összekapcsolása	60
3.6. Beágyazott SELECT parancsok	65
3.7. Al-SELECT használata más parancsokban	70
3.8. Másodlagos táblanevek	77
3.9. Indextáblák	77
3.10. Táblák törlése	77
3.11. SQL parancsok beágyazása programnyelvekbe	78
4. A HÁLÓS ADATBÁZISOK	85
4.1. A CODASYL DBTG javaslat	85
4.2. A hálós adatmodell alapelemei	85
4.3. Az adatbázis elemeinek megadási módja	89
4.4. Az adatbázis sémájának grafikus leírása	92
4.5. Az IDMS adatbáziskezelő rendszer	95

TITOKVÉDELEM SZÁMÍTÁSTECHNIKAI KÖRNYEZETBEN

A kiadvány az államtitok és szolgálati titok kérdéskörét érintő valamennyi jogszabályt egységes szerkezetben közli – segítséget nyújtva mindazoknak, akik számítógépen titkos adatokat kezelnek. A közzétett mintaszabályzatokkal a védelmi szabályzat előkészítőit támogatja. A titokvédelem eszközkínálatának termékismertetőjével a személyes vagy üzleti adatok biztonságos kezeléséhez is hozzájárul.

Ára: 150 Ft

Kecskés István ENGLISH FOR COMPUTER USERS

A számítástechnikában szinte létszükséglet az angol nyelv ismerete. A könyv alapvető célja, hogy hozzájáruljon a szakmai szövegek, problémák megfogalmazásához, továbbá olyan olvasási készséget fejlesszen ki, mely lehetővé teszi az angol nyelvű szakirodalom tanulmányozását és megértését.

Ára: 380 Ft



INFORMATIKA

Havonta

Hetente



Magyarország vezető
informatikai lapja.
Számítástechnika,
telekommunikáció,
irodatechnika együtt:
informatika – hetente
32 oldalon,
10 ezer példányban.



Magyarország vezető
számítástechnikai lapja.
Hardvertesztek,
hálózatok,
szoftverbemutatók,
alkalmazási tippek,
vírusinformációk –
havonta 100 oldalon,
10 ezer példányban.

CT PRESS

COMPUTERTECHNIK PRESS

H-1138 Budapest
Váci út 202.
Tel.: (36-1) 120-8007
Tel./fax: (36-1) 120-1636